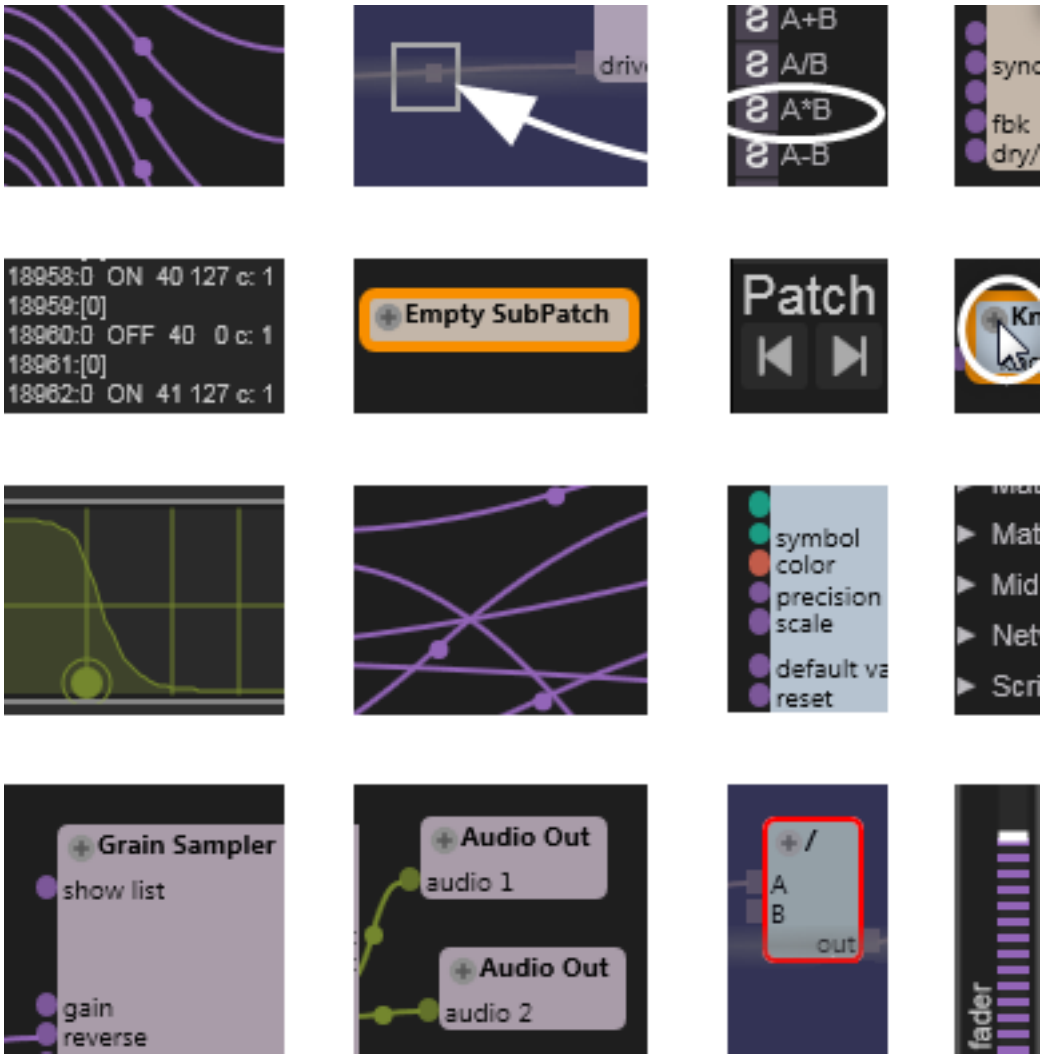


u2ine Hollyhock

Patching tutorial v 1.0

By **Yan Le Déan** and **Shane Hull**

Table of contents

Patching tutorial v 1.0 ..... **1**

1. Introduction **4**

2. Start-up Tips **5**

The browser search engine 5

The key shortcuts 5

Read the user's manual first 6

3. General Concepts **7**

Drag a module over a wire 7

Replace a module 7

Recreate module command 7

Do you need Faders or no Fader? 9

4. Monitoring and Analysis Tools for Diagnostics **11**

Monitoring Data's 11

Settings Panel *11*

Analysis tools 12

LED & Oscilloscope *12*

Flow Analysis *13*

Trace *14*

5. Flow Control Modules **15**

From 0 to 1 and from 1 to 0 15

Pass only if has changed 16

Pass/Stop flow 17

Toggle 19

6. Audio Patches **20**

Simple audio Effect 20

7. MIDI Patches **22**

Simple CC generator 22

Simple Note ON generator 23

MIDI transformations 23

Synchronization and the clock 24

8. Plug-ins Patches **25**

Simple interface 25

9. Transforming Data's **26**

Change scale 26

Values inversion 27

10. Sub Patches **28**

General concept 28

Add inlets and outlets to sub Patches 30

Data out module *31*

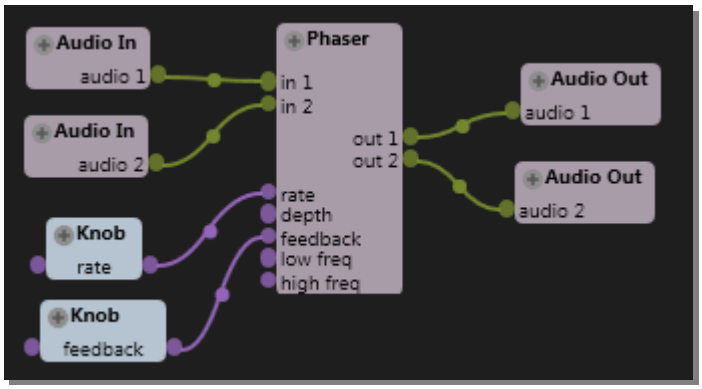
11. Polyphony for Sub-Patches **33**

Some polyphonic sub-patches examples 33

4in 4out Random Delay *33*
4in 4out Volume *34*
1in/4outputs Audio *37*
4 in 1 out Audio mixer *37*
Advanced Polyphony *37*
Apply polyphony and apply and reset 39
Apply the polyphony *39*
Apply and reset *39*
12. Math Modules 40
Arithmetic modules 40
Trunc / Round *40*
Multiplication *40*
Logic modules 41
Greater or Equal (>=) *41*
NOT *42*
AND *44*
13. Switches Groups 46
14. Control's Location 48
Container 48
Hierarchy of container *50*
Pop-up Window 51
15. Usine's Panels 52
16. Array's 53
17. Improving the CPU Load 54
Simplification *54*
Polyphony *54*
Patch ON/OFF *54*
Patch bypass *54*
18. Conclusion 55
Starting a project 55
Troubleshooting Problems 55

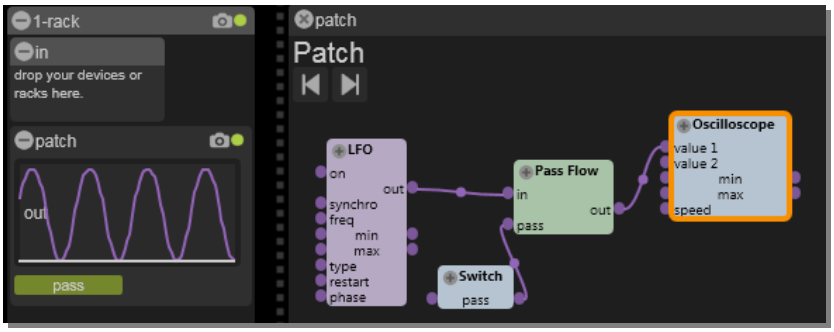
1. Introduction

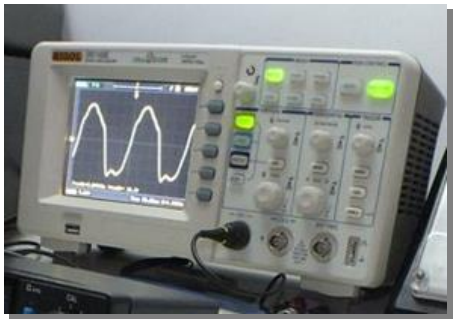
Patch, patching, in the music world, was originally used in the age of the first big modular synthesizers, where you could create patches, linking Oscillators to ADSR envelopes, Filter (LP, HP, BC and NOTCH) modules, and so on… Of course this technique is still available on many modern modular synthesizers.



So by modular we mean hardware or software groups that can be subdivided into other smaller groups called patches**.** Furthermore, these groups can also be built with sub-patches, with these ones being built from modules (the smallest basic elements in Usine). This is also the way you'll work in Usine Hollyhock.

In the same way, you can find parallelism between the world of electronics and Usine. Here's an example with a LFO (Low Frequency Oscillator), which creates a sinusoidal wave you can visualize with an Oscilloscope module in Usine. You can also add an electronic switch, here connected to a *Pass Flow* module, to pass or block the signal from the LFO to the Oscilloscope.





Before beginning these lessons, you should have read and assimilated the Usine Hollyhock user's Manual and practicing the examples included can be a good idea too.

Note: Hollyhock is always in evolution, so some colors and small details in this manual can be different from your version.

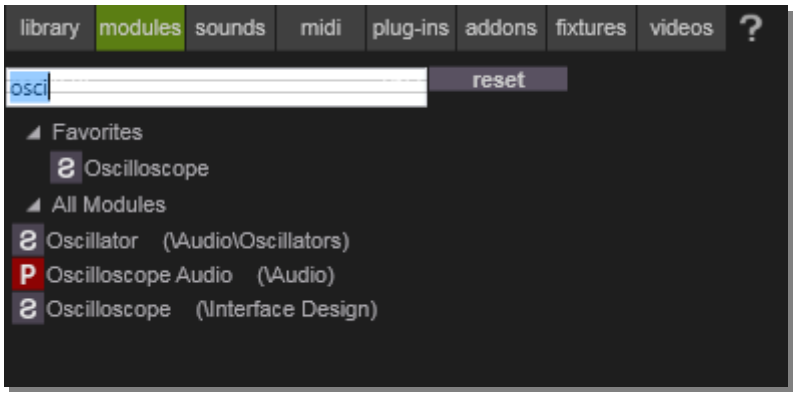
Current version: 2.0.081

2. Start-up Tips

The browser search engine

When patching there are two ways to find your modules.

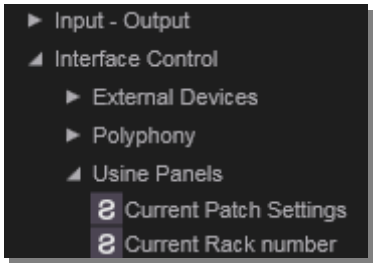
1) If you know the name or even only a part of the name of the module you're looking for, the quickest way is to use the search engine of the browser.



Note that that you can use a part of a word to find it. For example, if you are looking for *oscillators* just type *osci* in the search field of the browser.

Typing only *ob* will find modules like *knob* or *combobox*.

2) Of course you can also use the classic tree view of the browser to explore modules by categories



The key shortcuts

Another good habit is to learn the main shortcuts when patching. By using these shortcuts you will gain speed and efficiency.

Here are the main shortcuts you can use when patching:

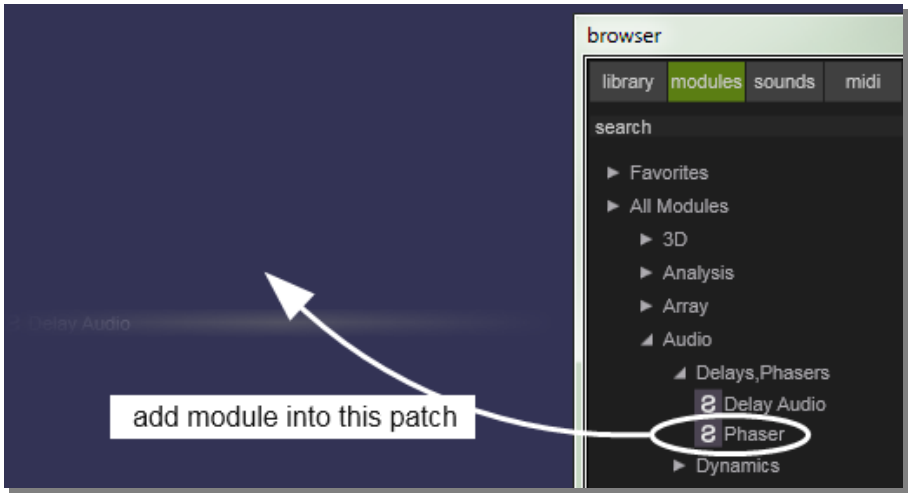
Show browser `[CTRL/CMD +B]`

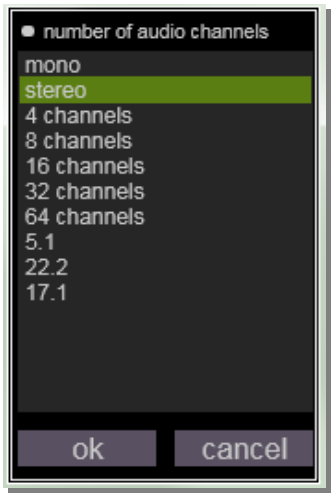
Edit design `[CTRL /CMD + D]`

Edit settings `[CTRL /CMD +I]`

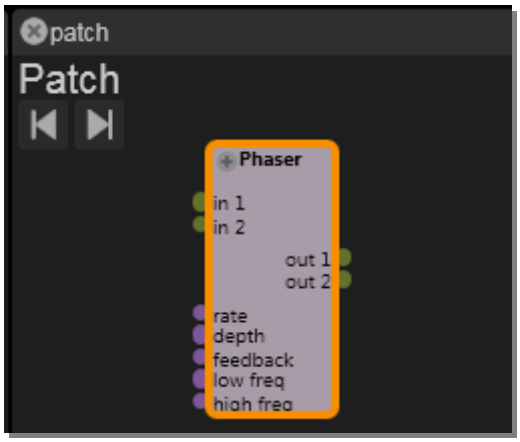
Read the user's manual first

Because we need, in this manual, to approach deeper concepts, we'll not explain the basic patches concept. For example on the user's manual, we show the drag & drop of each module when creating a patch. Here the *phaser* module in the stereo version.





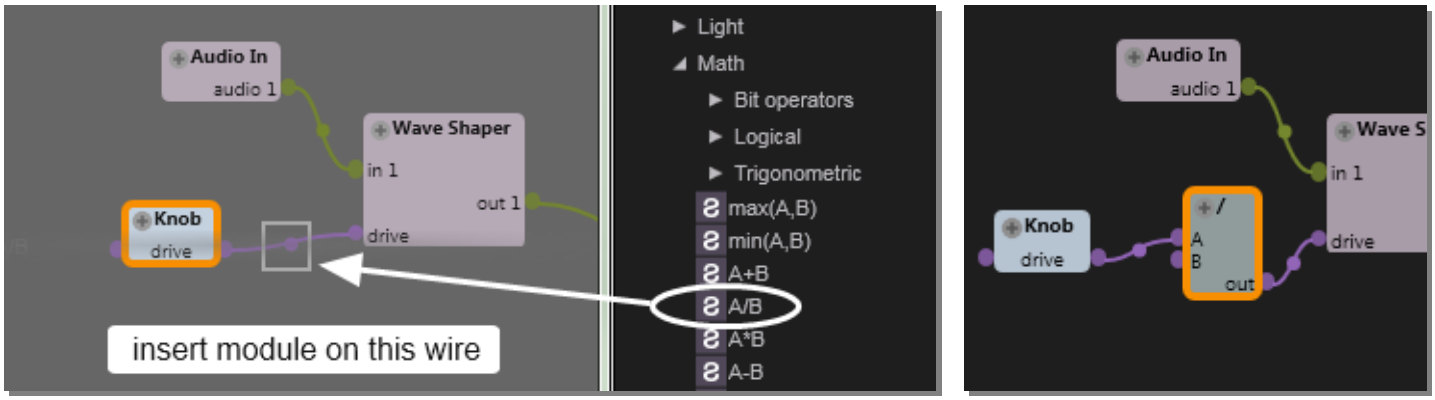
As you did user's manual exercises, you understood this. So, in this new manual, when we'll say *add a stereo phaser module*, you'll see directly:



3. General Concepts

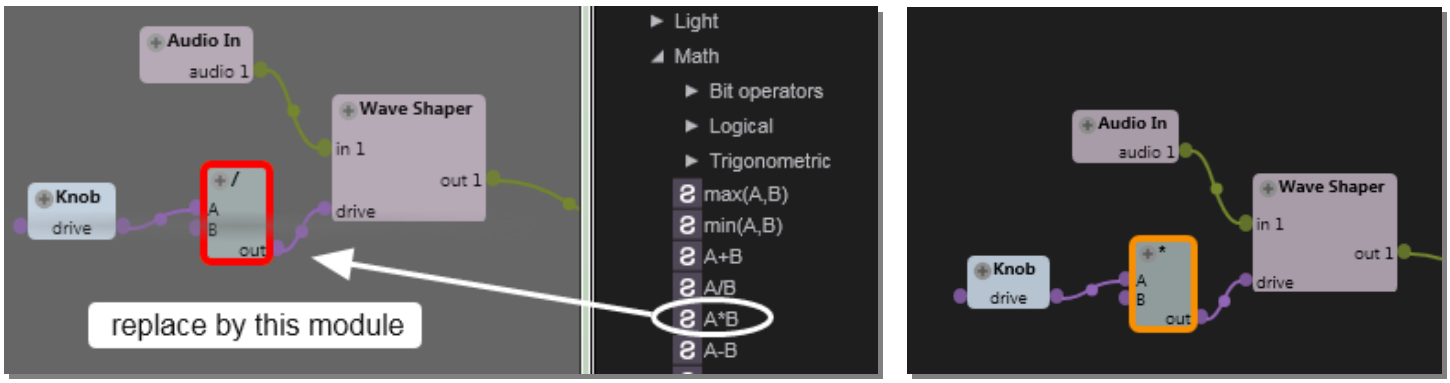
Drag a module over a wire

Usine offers an easy way to add a new module into the flow of a patch, simply by drag & drop a module on the square at the middle of a wire. This one will be selected and the new module will be automatically connected.



Replace a module

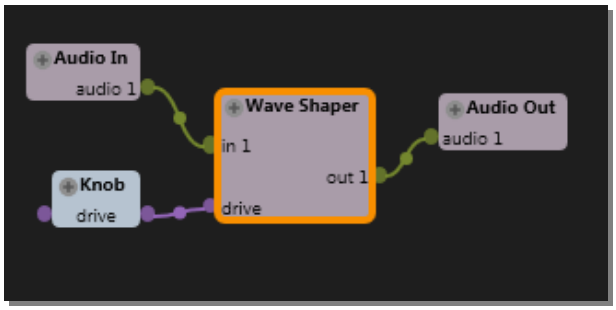
When creating a patch you can change your mind, sometimes because you make mistake or because you had found a better idea. In these cases, there is no need to delete the module. If they are similar in type (inlet types), simply drag & drop the new one over the old one while holding the alt key to replace it.

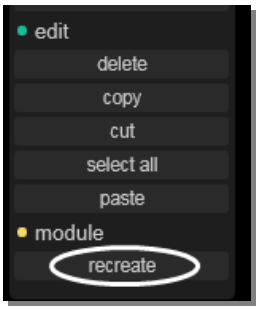


Recreate module command

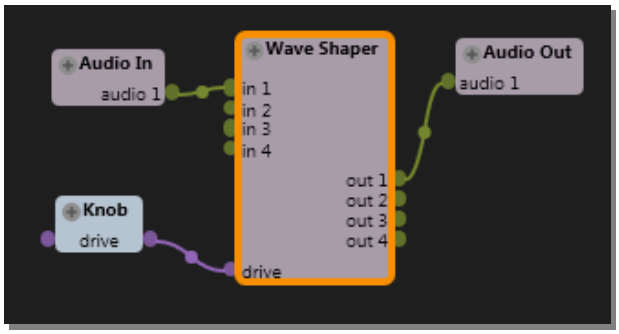
In Usine all audio modules can have up to 64 channels. Sometimes you will start a project with a stereo application in mind, and finally decide to turn it into a multi-channel (quadriphonic or more) project. In this case the recreate command in the contextual menu will prevent you from having to delete each module manually. This command will recreate the audio modules and allow you to revisit the initial inlet/outlet menus of each module.

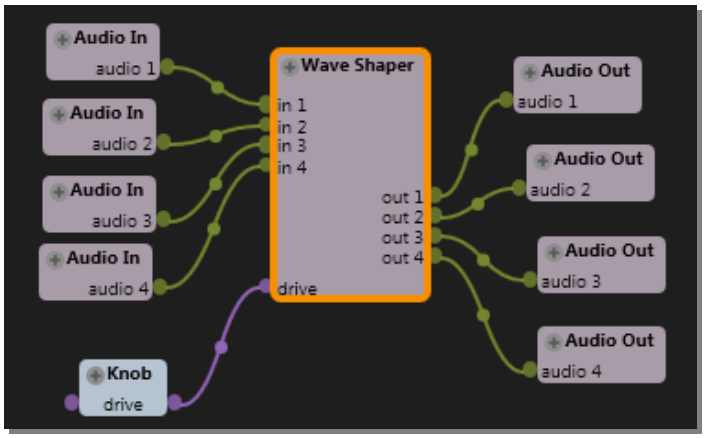
Here is an example with a stereo *wave shaper* module. We use the recreate command in the contextual menu and choose 4 channels in the pop up window:



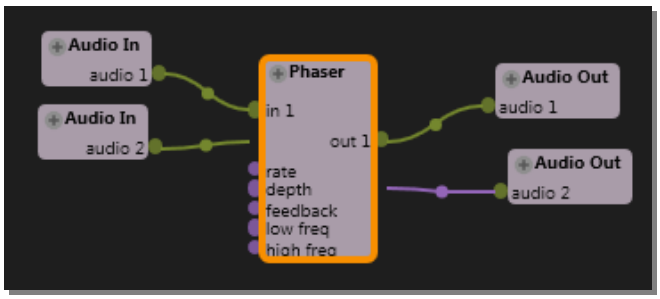


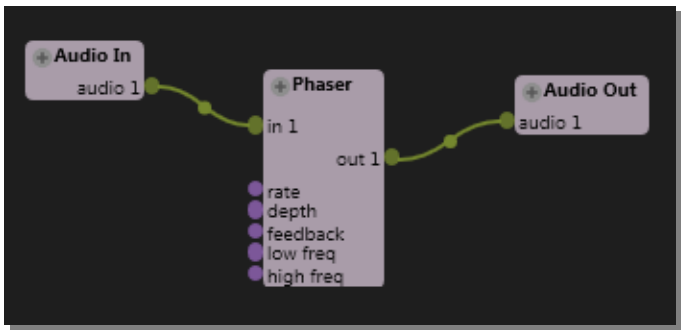
The *wave shaper* module is recreated, but this time with 4 inputs and outputs. You'll only need to create the new inputs/outputs:





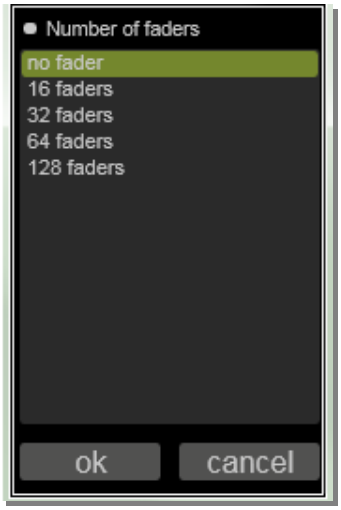
Of course if you use this option to decrease inlets/outlets some errors can appear. But don't worry, simply delete the unnecessary entries and outputs modules. Here, with a *stereo phaser* module that we have recreated for a mono version, an inlet and an outlet need to be deleted:



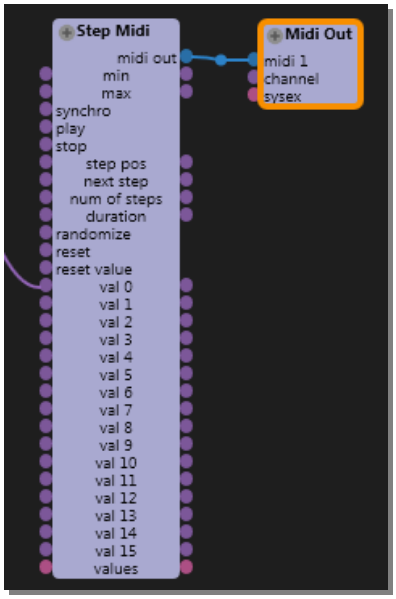


Do you need Faders or no Fader?

When creating modules like array's, step lines or step MIDI, you will see a dialog box which will ask you if you need *no fader* or N faders

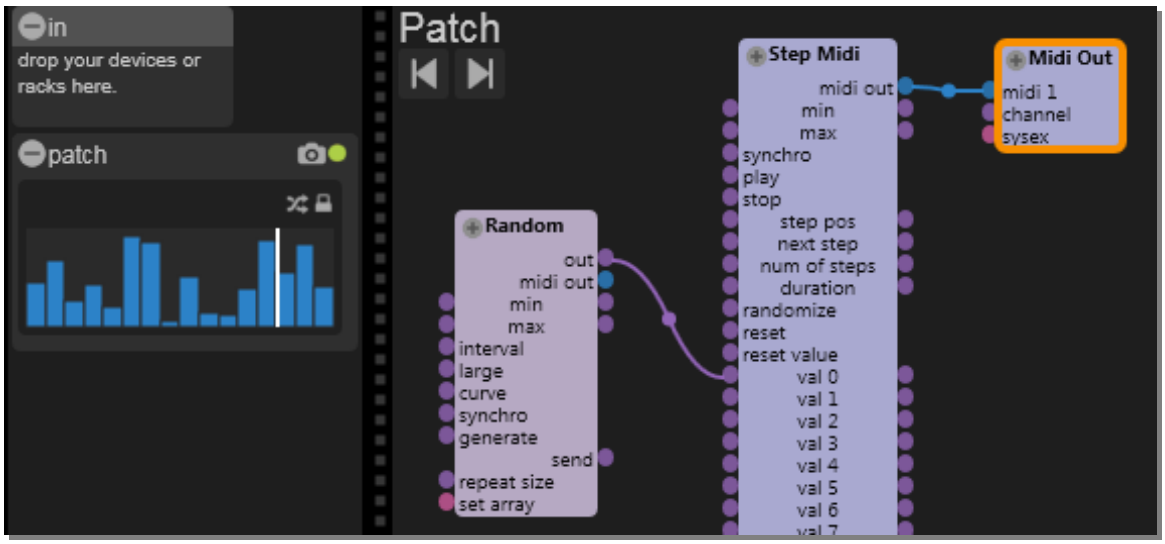


If you choose 16 faders when creating the *step MIDI* module, 16 inlets/outlets will be visible which correspond to the first 16 steps.

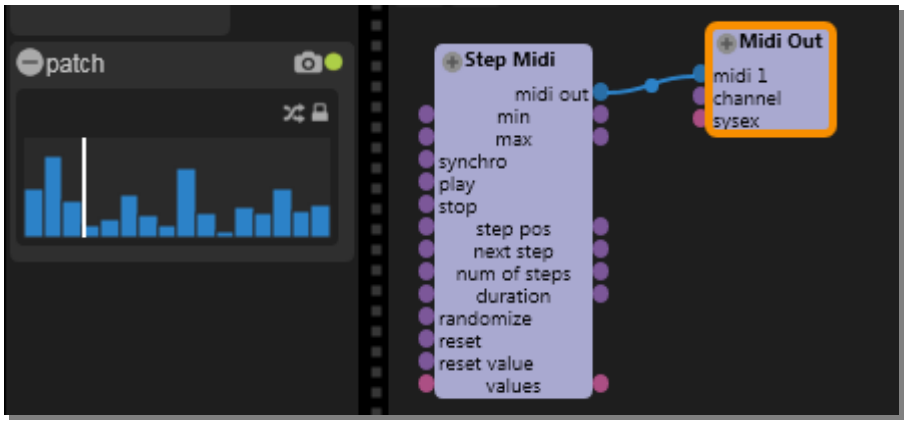


This will be useful if you want to modify steps values individually in the patch from an external fader or design object.

As an example below, we added a random generator module to the first step of the step sequencer.



Here you will see that we have created a *step MIDI* module with the option “no fader". As you can see, no inlets/outlets are visible on the module itself.



4. Monitoring and Analysis Tools for Diagnostics

Monitoring Data's

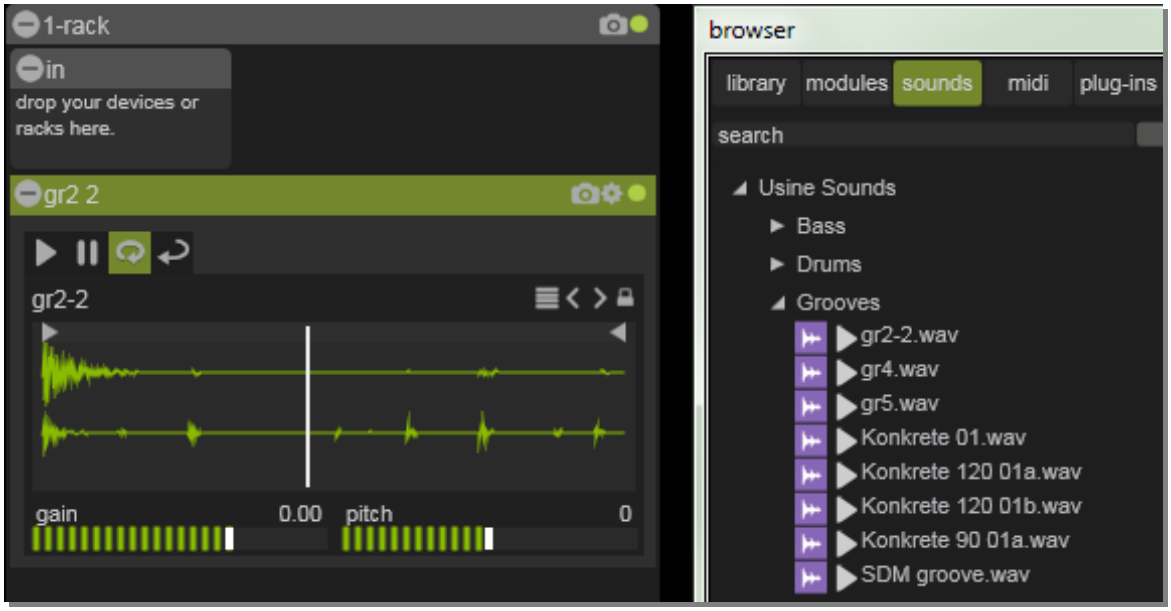
Usine provides a large number of tools for monitoring and understanding data as it passes through the wires. These are helpful to debug and understand how modules or patches work.

Settings Panel

The basic information tool in Usine is the settings panel, which appears when you right click on a module. The settings panel is used to show the status of a module and to configure its properties.

The type of information displayed will vary from module-to-module depending on its function. However there are a number of common displayed elements that you will encounter.

First, drag & drop a wave file from the browser into an empty patch.



Now, double click on the patch you have just created to open it's structure and select the *sampler* module to see its settings panel.

Observe the panel:

The *is playing* LED will light, showing that a sample is playing and, each time the sample starts again, the *re-starting* led lights up briefly.

Finally, have a look the *pos* (position) in the range 0… 1. It provides both graphical status and detailed numeric values. (0, 11505 in this example).

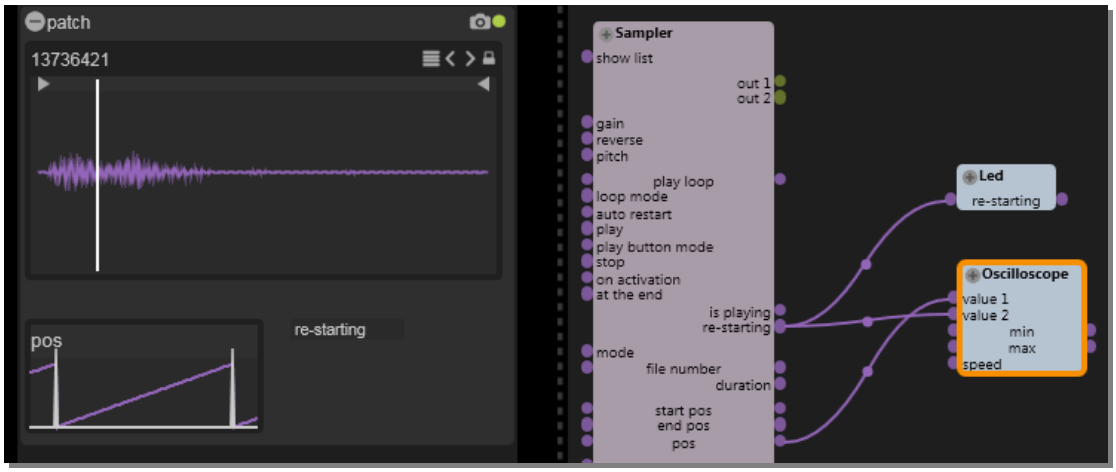
Analysis tools

Usine provides several modules to analyse and understand what's happening when an action is executed.

LED & Oscilloscope

From the sampler module, click & drag a wire from the outlet *pos* (position) and choose *oscilloscope data* in the menu. Link now the restarting outlet to the *value 2* inlet of the Oscilloscope module.

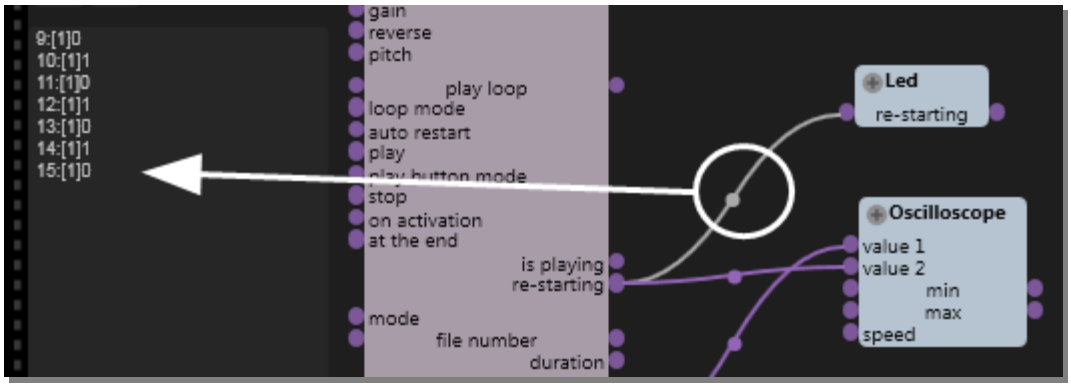
Use the same method to add a *Led* module from the *re-starting* outlet:



You can also link the *restarting outlet* to the *value 2* inlet of the *oscilloscope*. Now, you have a better visualization of what's happening:

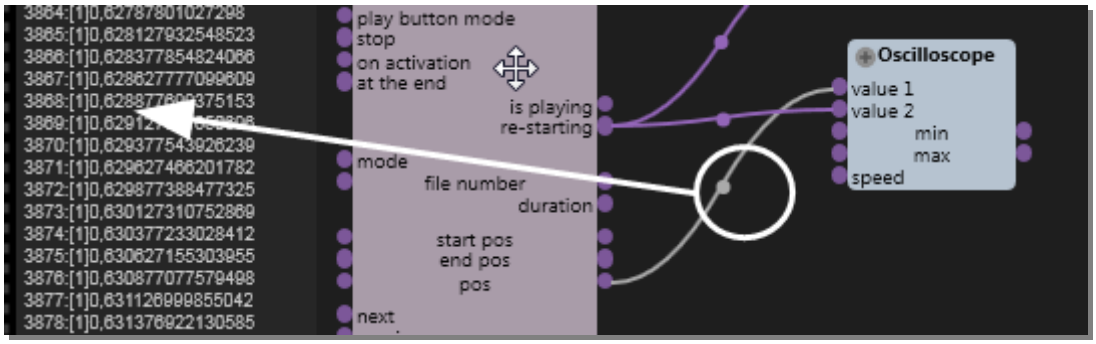
- A brief pulse from the outlet *re-starting* (white).
- A regular progression from the outlet *pos* (position) (purple).

An alternative method exists when you click on the wire from the *re-starting* outlet. This will open a console on the left part of the patch**:**



In this way, you'll see that, each time the sample starts, a brief pulse of **1** is sent, then back to the **0** value. It's called a trigger.

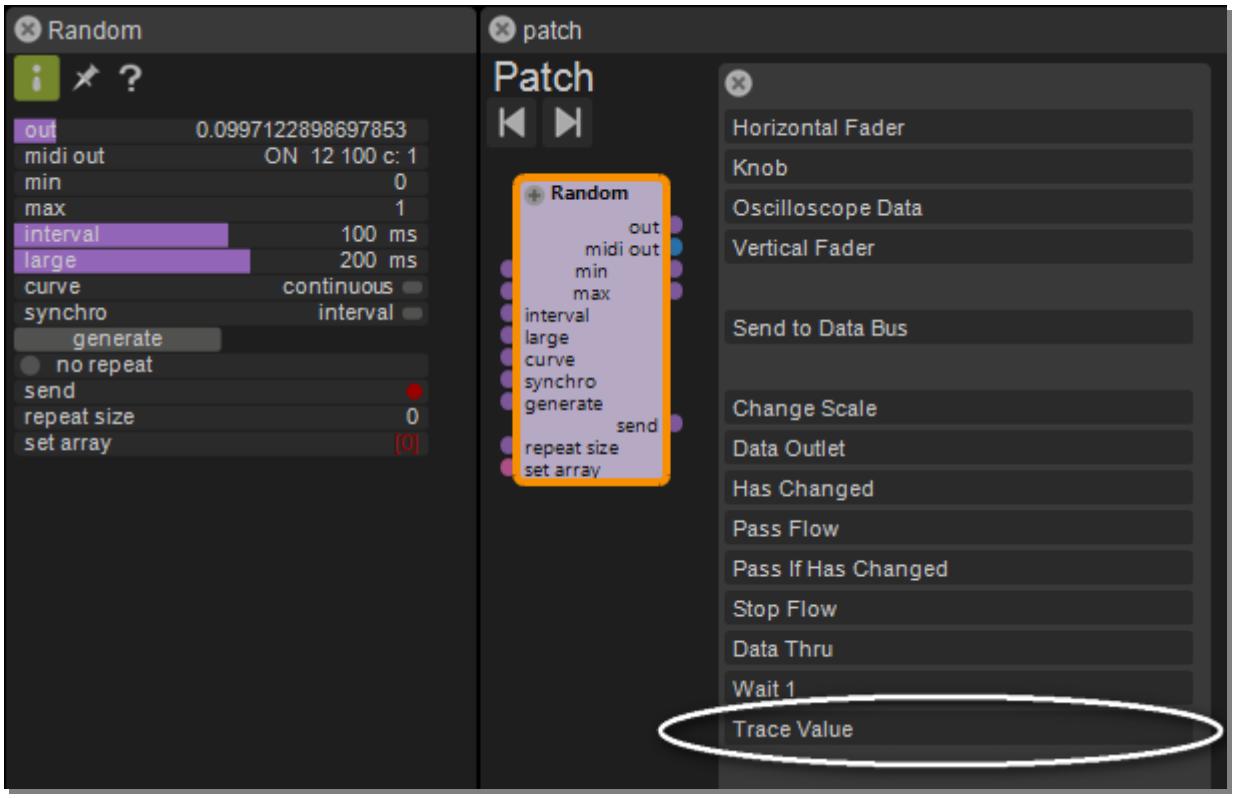
If you use the same method from the *pos* outlet you'll see continuous values from **0** to **1** fitting to the position in % of the sample. *You can stop the audio engine [SPACE] to see the progression more easily.*



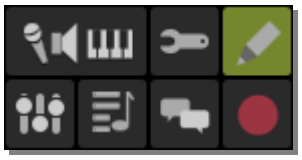
Trace

You can use the *trace* module to display flow contents in the trace panel.

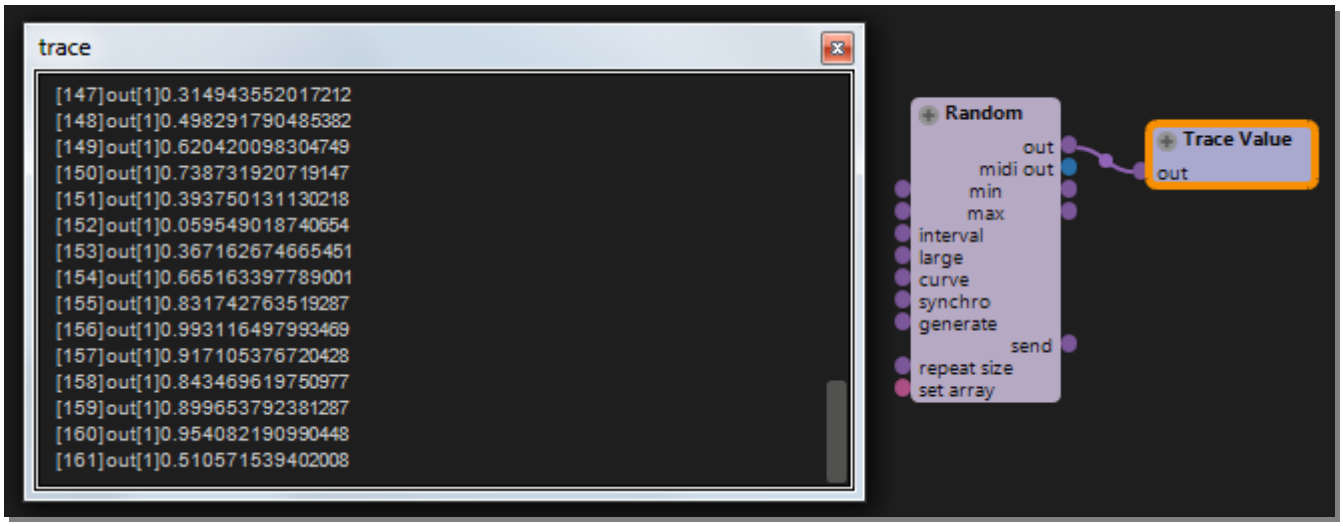
For example, add a *Random data* module into a patch. Then Drag a wire from the *Random/out* and choose the template *Trace Value*:



Open the trace panel by clicking on the trace switch



A window will show the following:



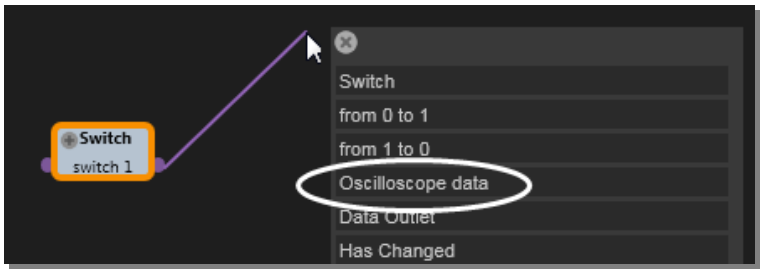
Outputs values of the *Random* module are displayed in the trace panel.

5. Flow Control Modules

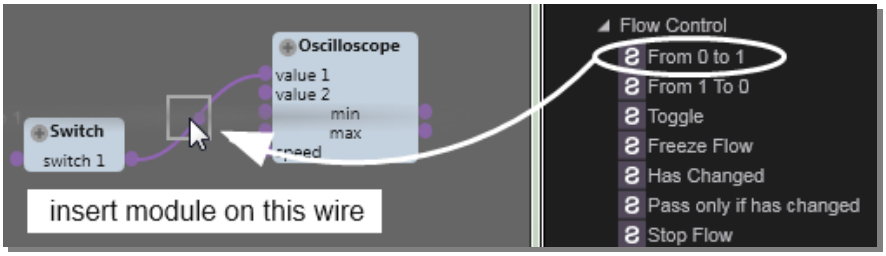
In the **Flow Controls** modules folder of the library, you will find very useful tools to handle signal flow in several different ways.

From 0 to 1 and from 1 to 0

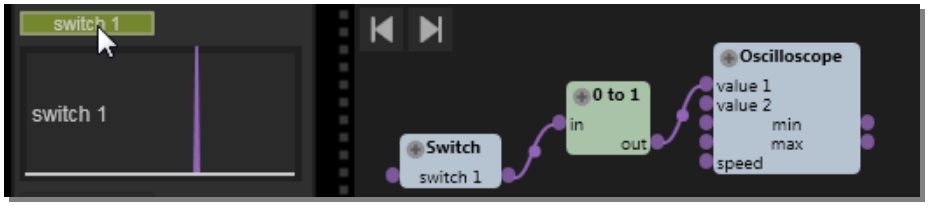
To understand the behaviour of this module, we will first add an *oscilloscope* module to the switch.



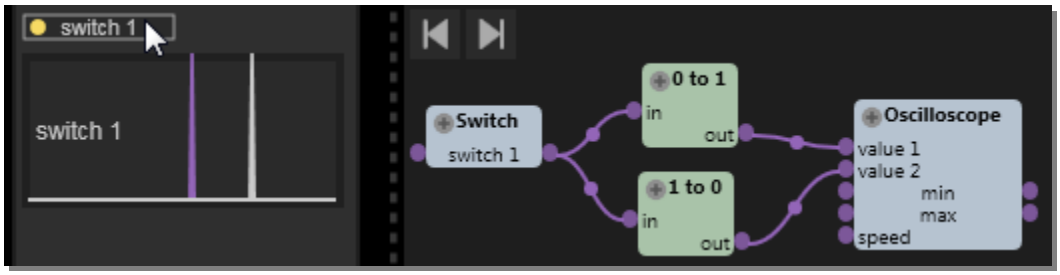
Once we have created the oscilloscope, drag & drop a *From 0 to 1* module onto the wire.



You can now see every time the switch is set to ON (when the value of the switch changes from 0 to 1), a quick impulse (drawn in purple) is sent to the oscilloscope.



Again, connect the switch to the *oscilloscope*, but this time use the *value 2* inlet. Then add a *From 1 to 0* module. The oscilloscope will react once the state of the switch changes and sends an impulse. The first one does it when the value goes from 0 to 1 (purple), the second one when the value goes from 1 to 0 (white).

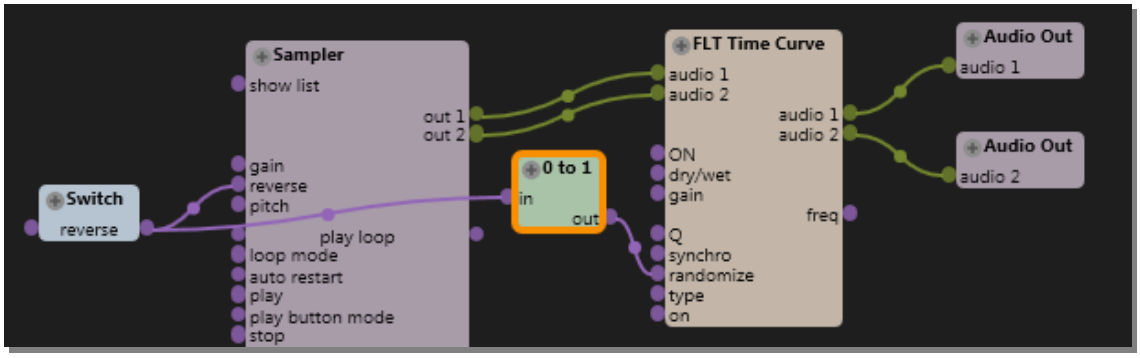


Case study

I would like to achieve that, each time I use my reverse switch on my sampler, a new random value is sent to a Flt Time curve sub-patch (which you may find in the browser/library/filters tab).

Method

Add a *From 0 to 1* module to the out of the *reverse* switch of the sampler and connect the outlet to the *random* inlet of the *flt time curve* sub-patch.



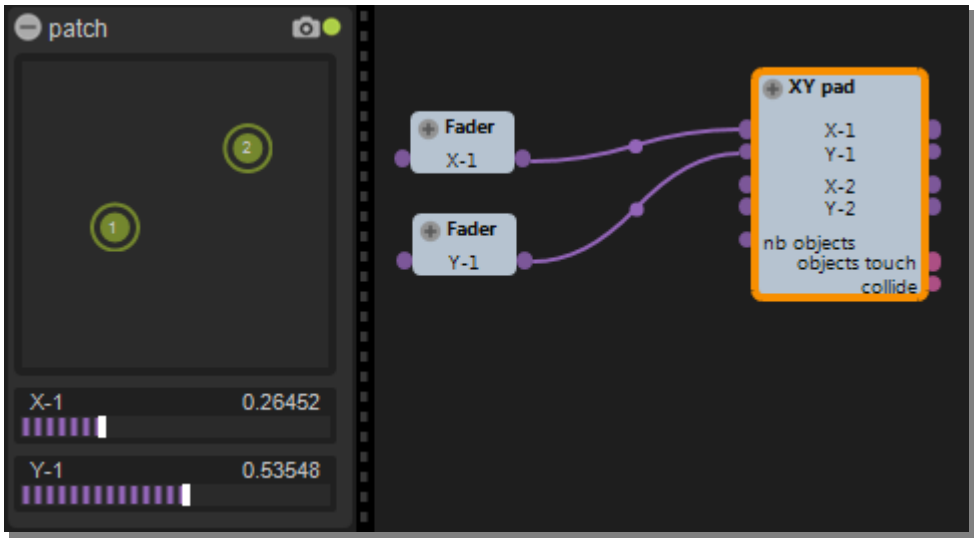
Pass only if has changed

Case study

I use a XY pad and I want to be able to change the balls positions with the mouse or with faders.

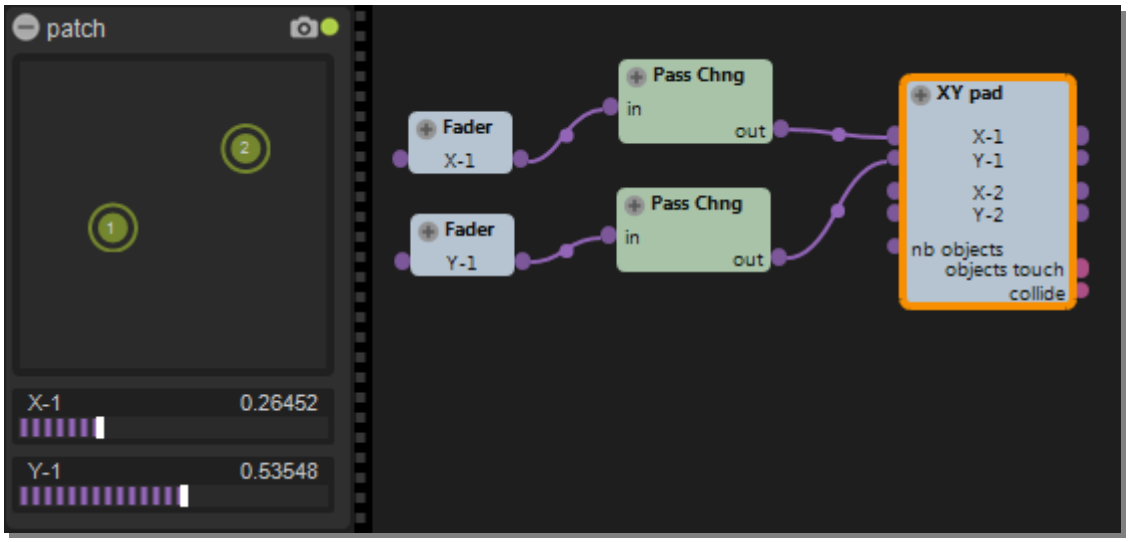
Method

Add a *XY Pad* module and two faders to obtain the following patch:



The main problem is that once the faders have been connected, it is impossible to move the ball 1 with the mouse because the faders drive the position.

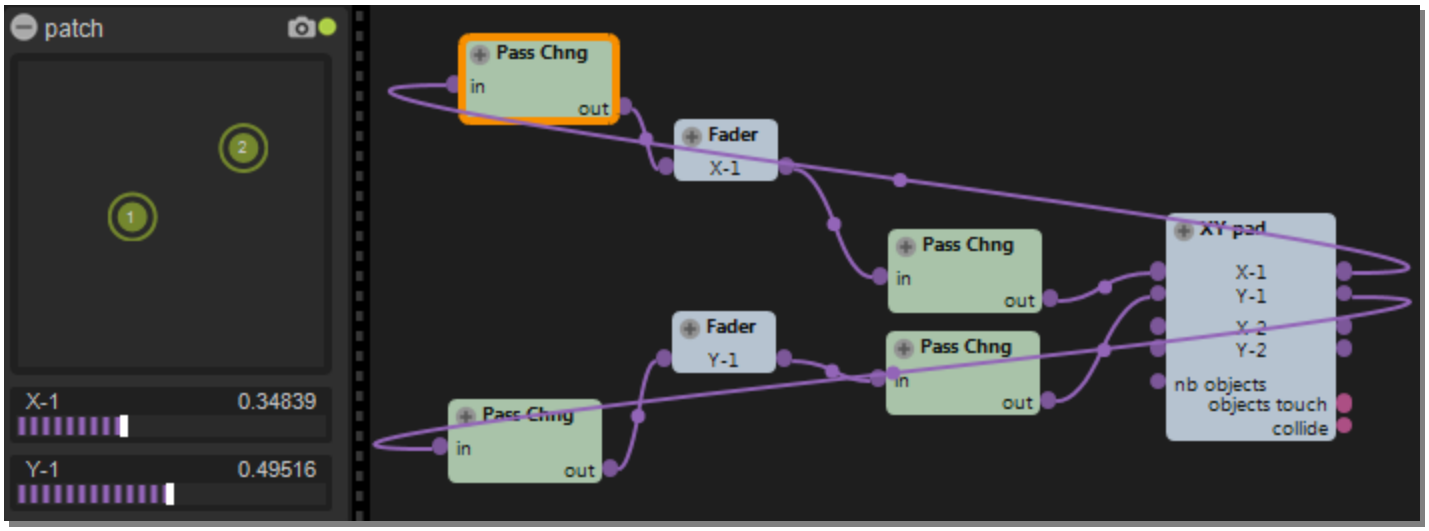
The solution is to use *pass only if has changed* modules and insert them on the wires as shown:



The signal now only passes if the *fader's* values have changed. If the fader's value does not change, the wire will act like it does not exist. With this configuration you can modify the ball 1 position with the mouse, a tactile screen or with faders. The *pass only if changed* module lets the signal pass the value only if it has changed.

A small problem remains. Now, if you change the Ball 1 position the position of fader is not updated to reflect the real ball 1 position.

To solve this, just add two more *pass only if has changed* modules and connect them as follows:



Now the fader's values reflect the position of the ball 1.

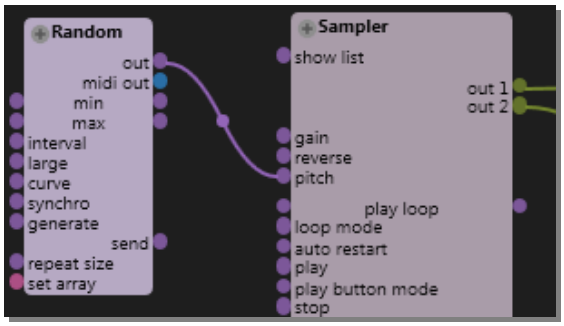
Pass/Stop flow

Case study

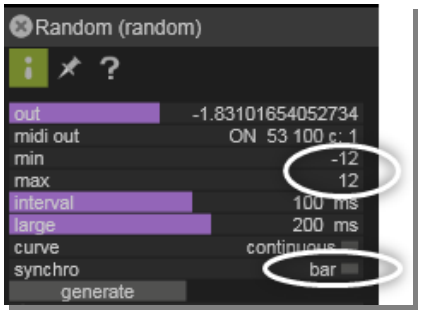
I want to send random data to the pitch inlet of my sampler, but I also want to be able to stop this flux. Then, every time I stop the flux the pitch is reset to 0.

Method

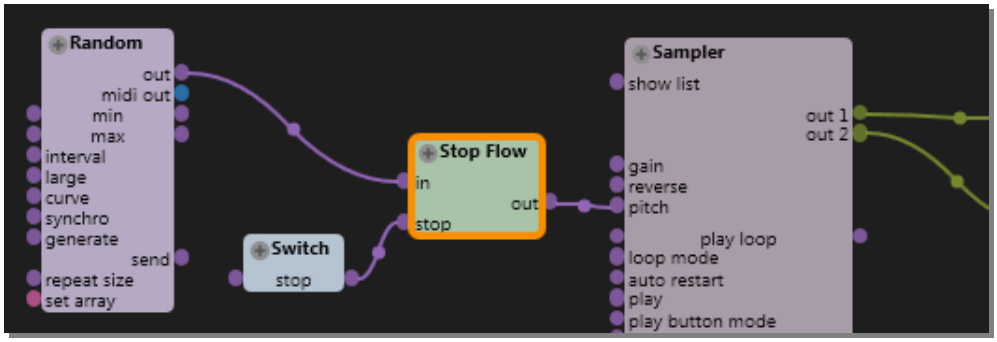
Add a *random generator* module:



For this exercise, set the minimum to -12, max to 12, and *synchro* on *Bar* in the *random* module.



Add now a *stop flow* module on the wire and a switch on the *pass* inlet.

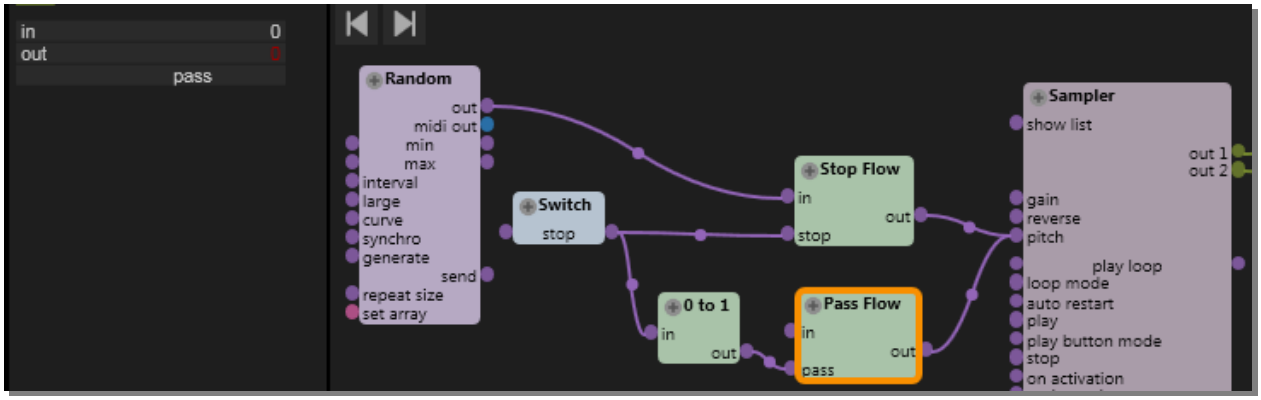


The *stop flow* module acts like an electric switch: when its *stop* inlet receives the 1 value then the flux is stopped.

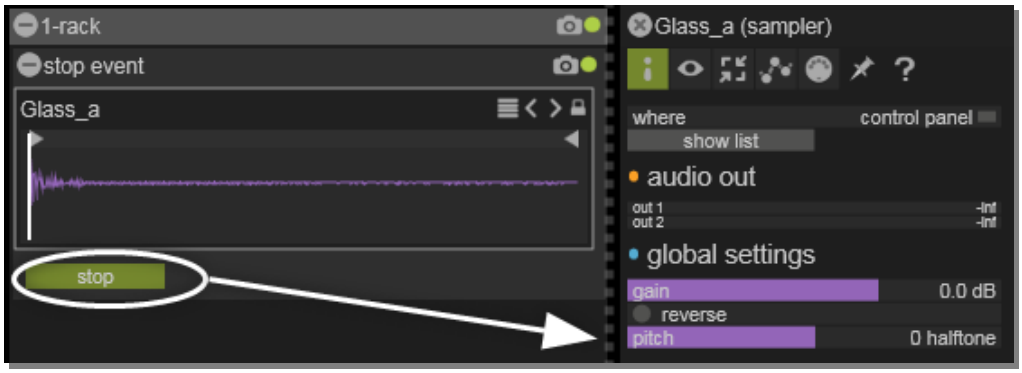
To resolve the second part of the question, "pitch back to 0 when the flux is stopped", just add two modules:

1. *pass flow* module which the flux then passes when it receives 1,
2. *0 to 1* module which sends a brief impulse when the switch is OFF.

By default the *pass flow* module has a 0 value in its *in* inlet, good news because it's exactly the value we want to send to reset the pitch.



The switch has now 2 actions: it stops the *random generator* flux and at the same time, sends a brief impulse to reset the pitch to 0:

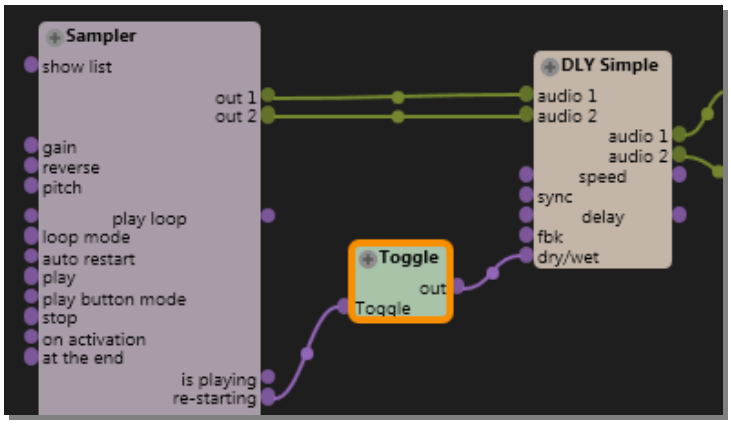


Toggle

The Toggle module alternates from 0 to 1 (values by default) each time the *toggle* inlet receives a 1 value.

To illustrate this module link the restarting outlet of the *sampler* module to the *toggle* inlet.

Then, link the out of the *toggle* module to the *input* of the DLY Simple patch dry/wet inlet

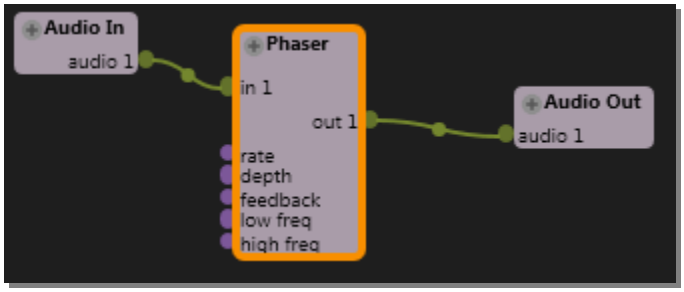


Now the *sampler* will play with a delay effect once every two sample restarts.

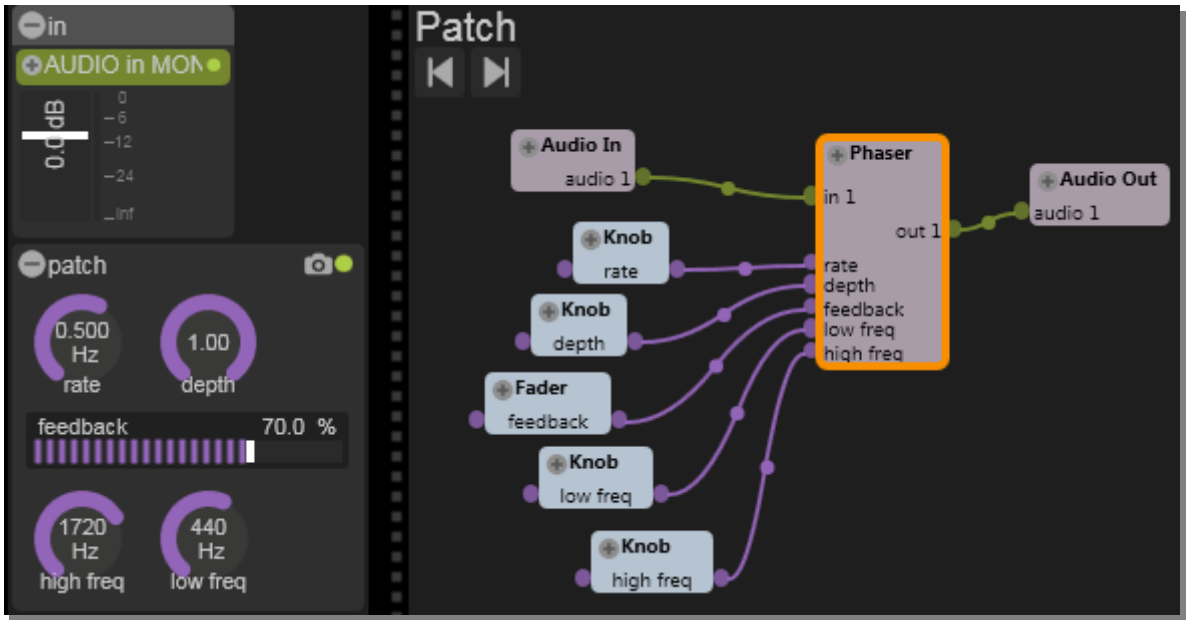
6. Audio Patches

Simple audio Effect

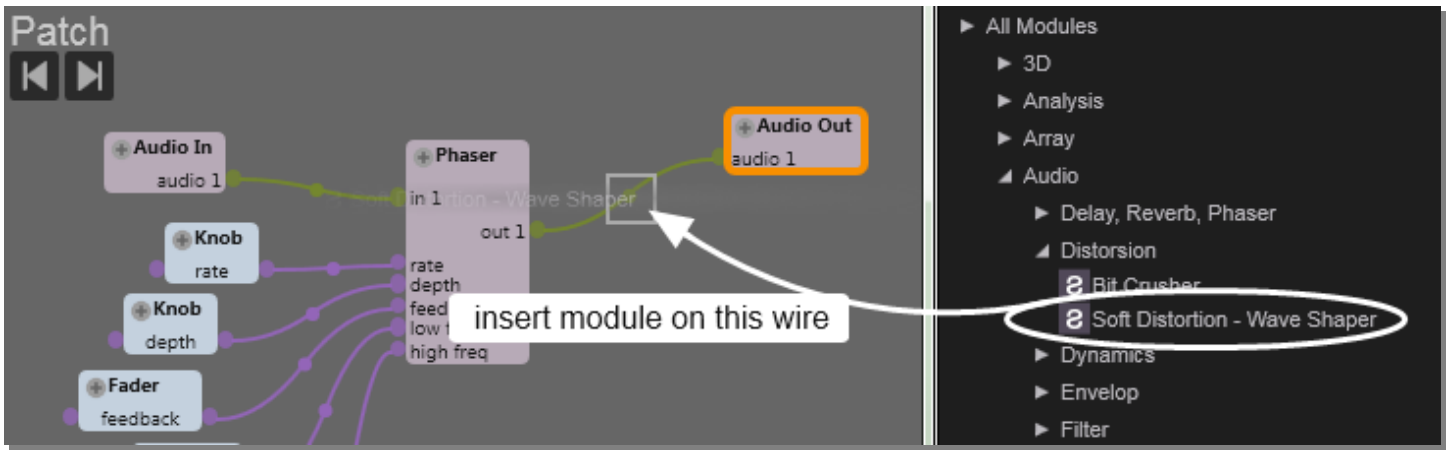
The Usine library provides several ready to use audio modules. For example a simple *phaser* effect, here in a mono version connected to audio in and out modules.



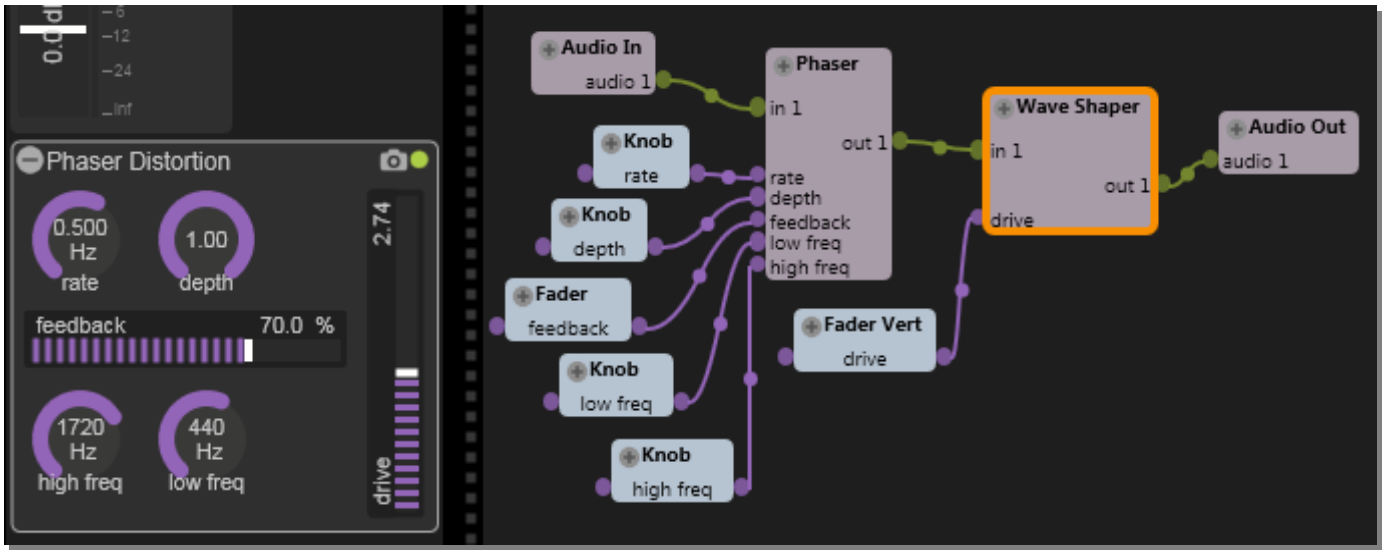
You can create all modules from its inlets to complete the patch in a minute



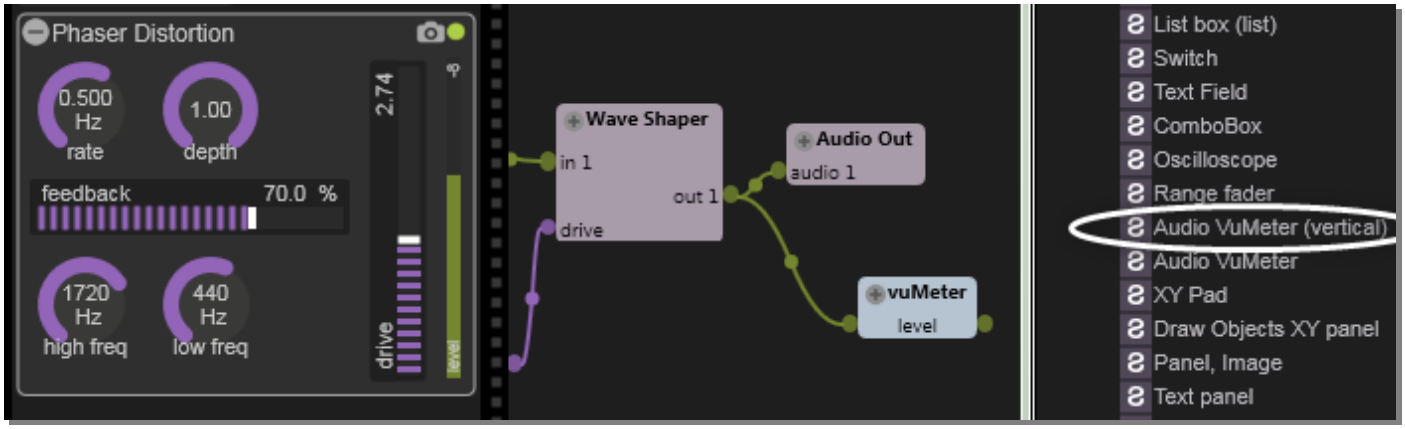
Now you can route the *phaser* to a *wave shaper* distortion. Simply drag & drop a *soft distortion* module over the wire to add it:



Add a vertical fader to the drive inlet and you will have a totally new and creative effect very quickly.



To improve the patch, you can also add an output vu-meter:



7. MIDI Patches

Before, take a look on the MIDI Implementation wiki page to understand elementary MIDI specifications.

Let's see two simple examples to send Control Changes (CC) and Note ON/ OFF messages.

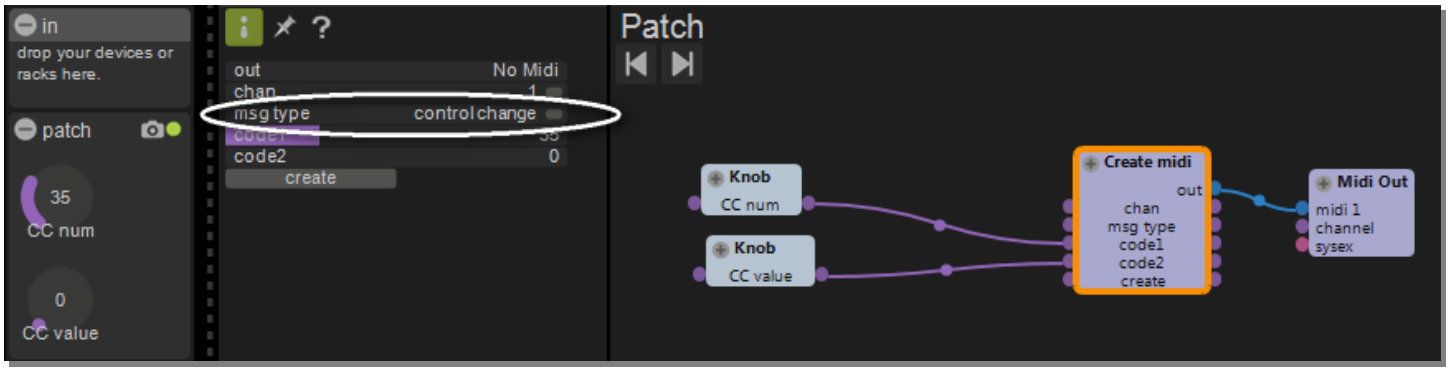
Simple CC generator

Sometimes it can be interesting to control a Plug in or MIDI hardware's as a MIDI synthesizer with Usine, for example on a tactile surface.

We use a *Create Midi message* module. Module's parameters are:

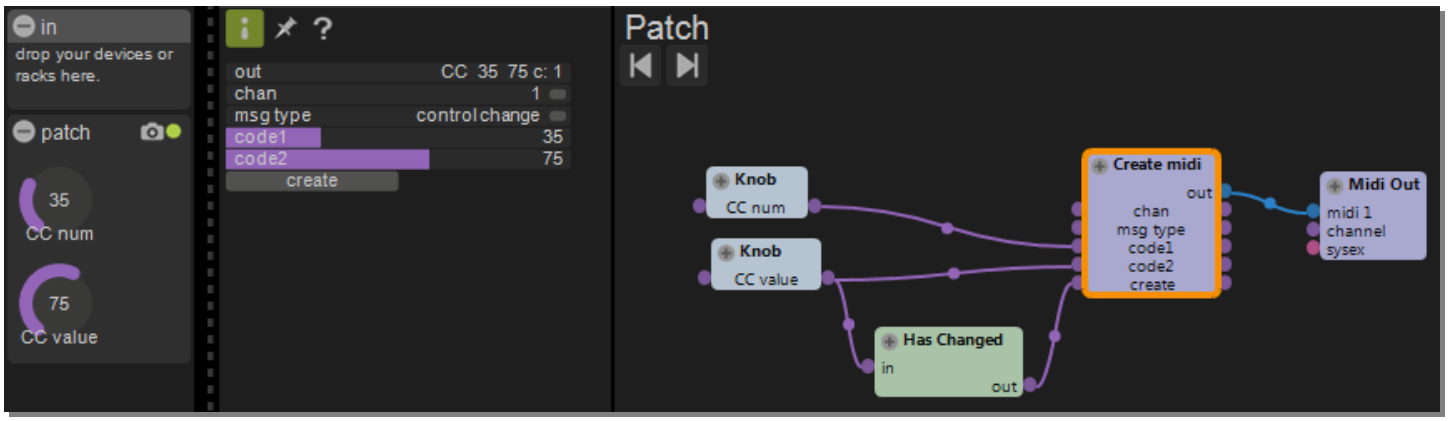
- **out:** MIDI flow out,
- **chan:** Channel of the MIDI message,
- **msg type:** Message type we can send (note ON, Control Change, etc.),
- **code1:** First data of the MIDI message (it will be the Control Change number we want use, see MIDI Implementation),
- **code2:** Second data of the MIDI message (it will be the value of the Control Change, see MIDI Implementation),
- **create:** A new message is created each time the button is triggered.

Create the following patch with a *Create Midi message*, a *Midi output* and two *knobs* modules:



In the *Create Midi* module, select *control change* in the *msg typ* combobox.

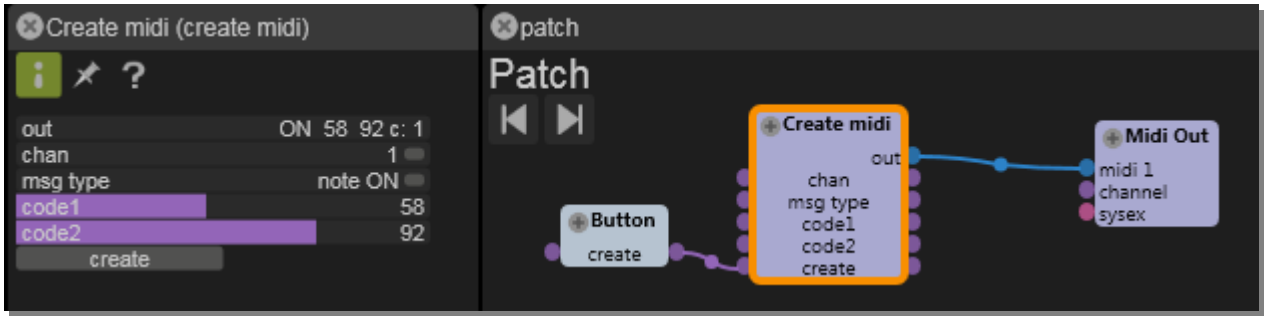
Now we would like to create a MIDI message each time the *CC value* knob is modified. For that we can use a *Has Changed* module which sends a trigger each time the input value is modified. Add it on the patch and connect it as follows:



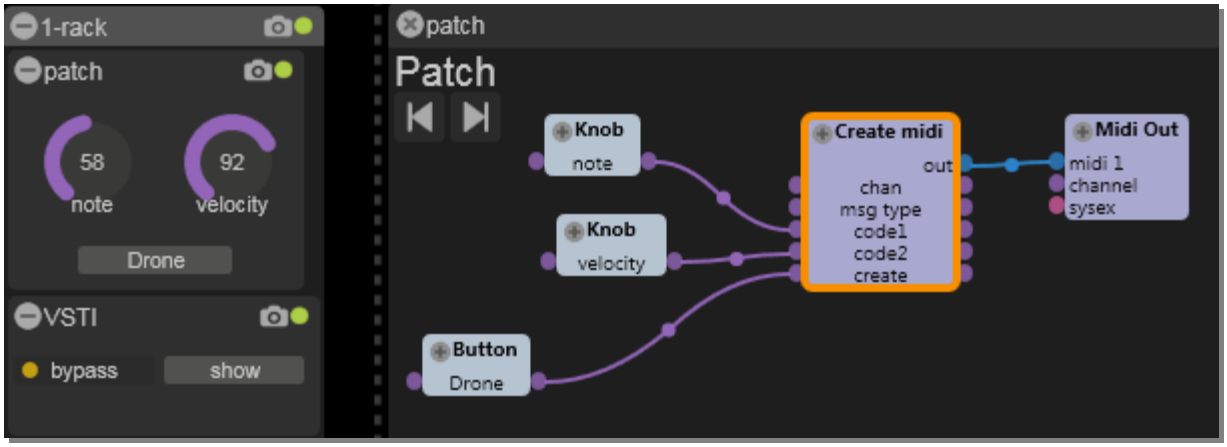
Now the patch works has expected: each time the *CC value* is modified it passes a CC MIDI message.

Simple Note ON generator

Let's create a simple drone. A note ON message is sent, no note OFF. Again, *create a Midi* module. Assign the **code 1** value to the Note number and the **code 2** the Velocity value. A *button* module will send 1 value (trigger) to create the MIDI message.



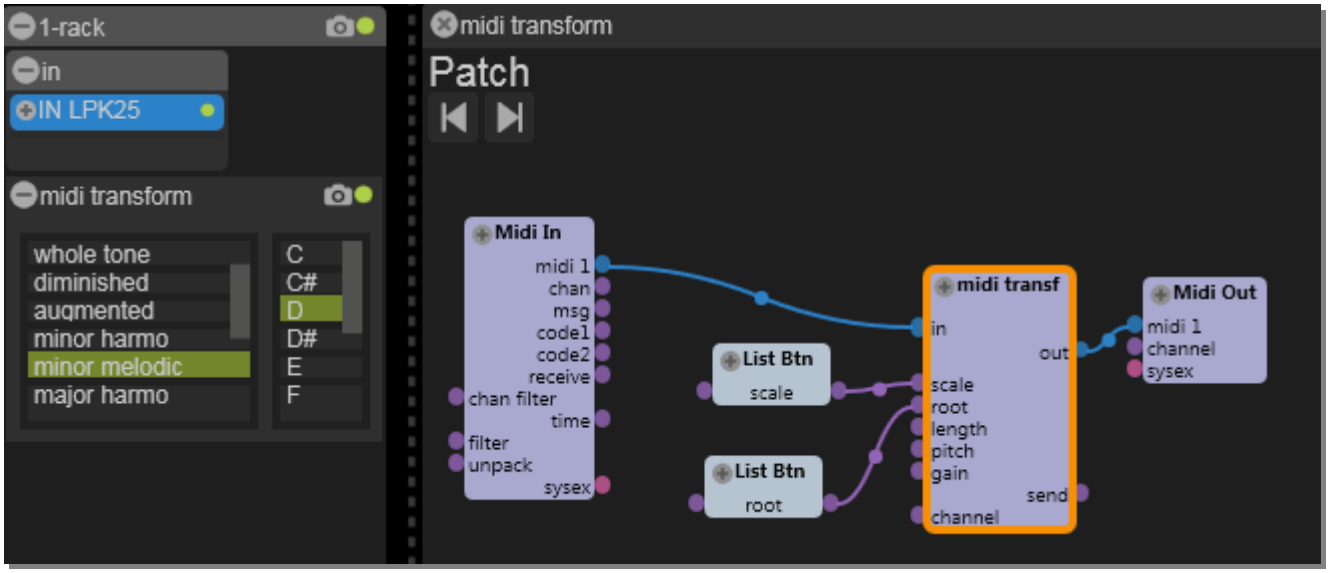
Add two *knobs* with the min and max values from (0...127). The first knob is the Note number connected to code1, and the second is the Velocity. The two *knobs* and the *button* now can be renamed.



MIDI transformations

Another important module is the *Midi transformer. The Midi transformer* lets you change the scale and the root of a MIDI flow in real-time.

Once inserted between the Midi in and Midi Out modules create two *listboxes* from its scale and root inlets to obtain:



Synchronization and the clock

Now we can imagine we want sequence these roots changes, 2 bars in A, 2 bars in D … to create some interesting progressions.

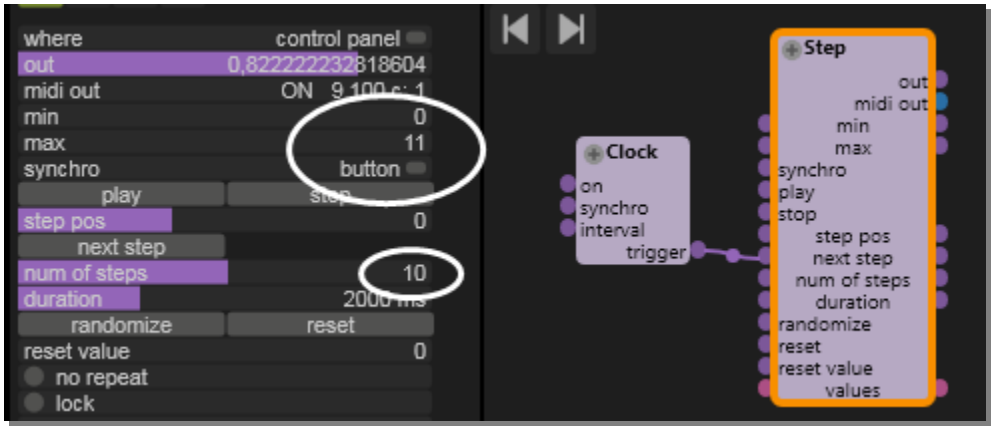
You first need to remember that all list modules in Usine start at the index 0. It means that the first value will be 0, then 1, then 2, etc. We can apply this to the scale *list buttons* module. In this way the first value (0) will be the root C, second C # and so.

Here a small table as reminder:

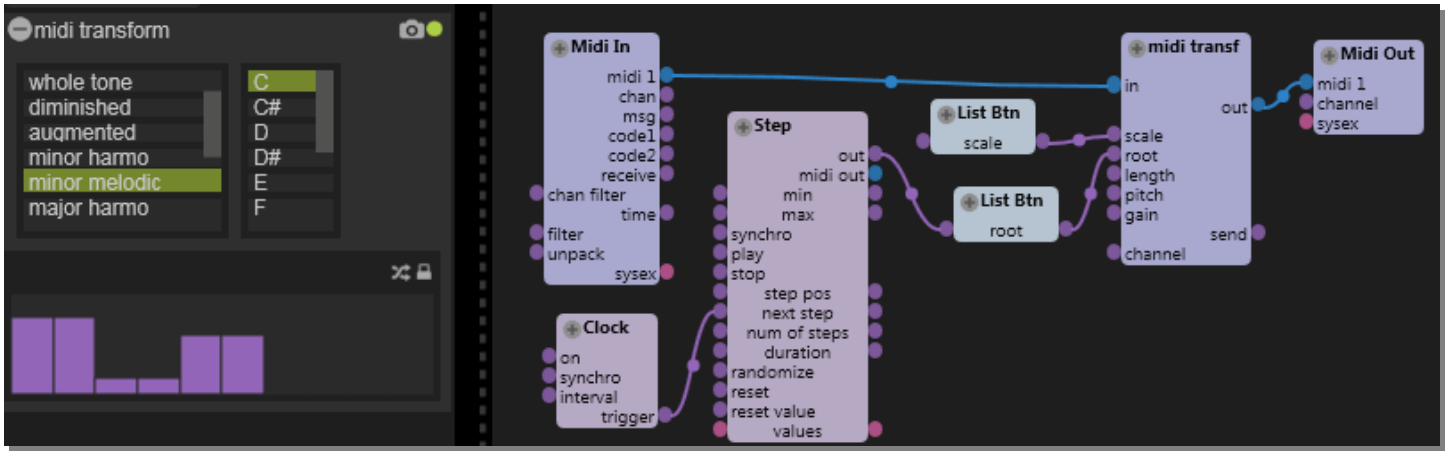
0=C	6=F#
1=C#	7=G
2=D	8=G#
3=D#	9=A
4=E	10=A#
5=F	11=B

To drive the sequence, we'll use a *Sequenced steps* module, minimum (min) and maximum (max) values are set respectively to 0 and 11. We have 12 possible values corresponding to the roots. Its synchronization (synchro) is set to *button*.

So we can drive the *next step* inlet with a *clock* module that offers several choices for synchronization, for example *bar*.



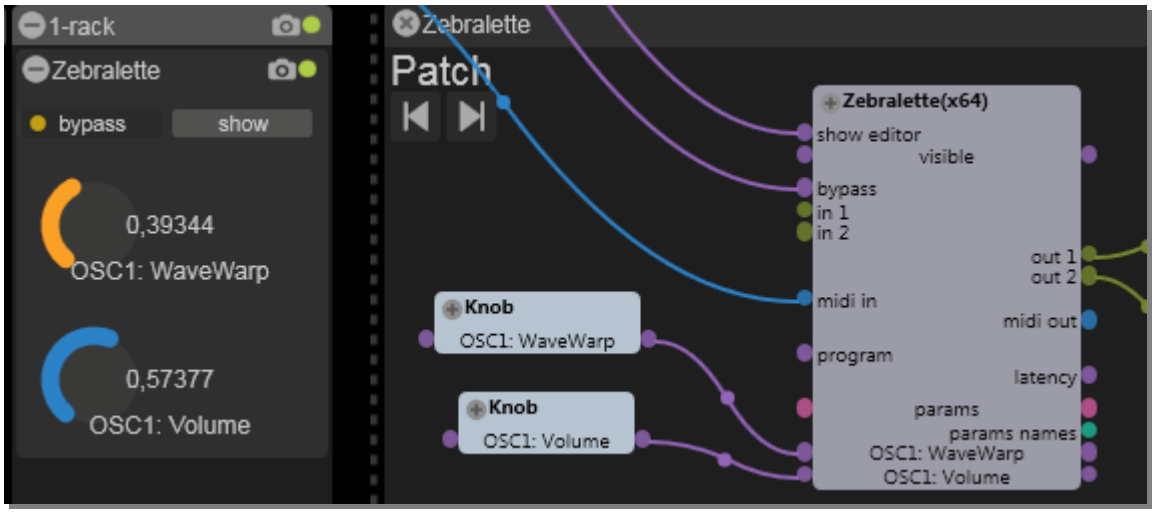
Now we can link the *Sequenced Steps* to the root *listbox* and choose a progression, let say 9/2/7/0 values. It will give us a A/D/G/C progression:



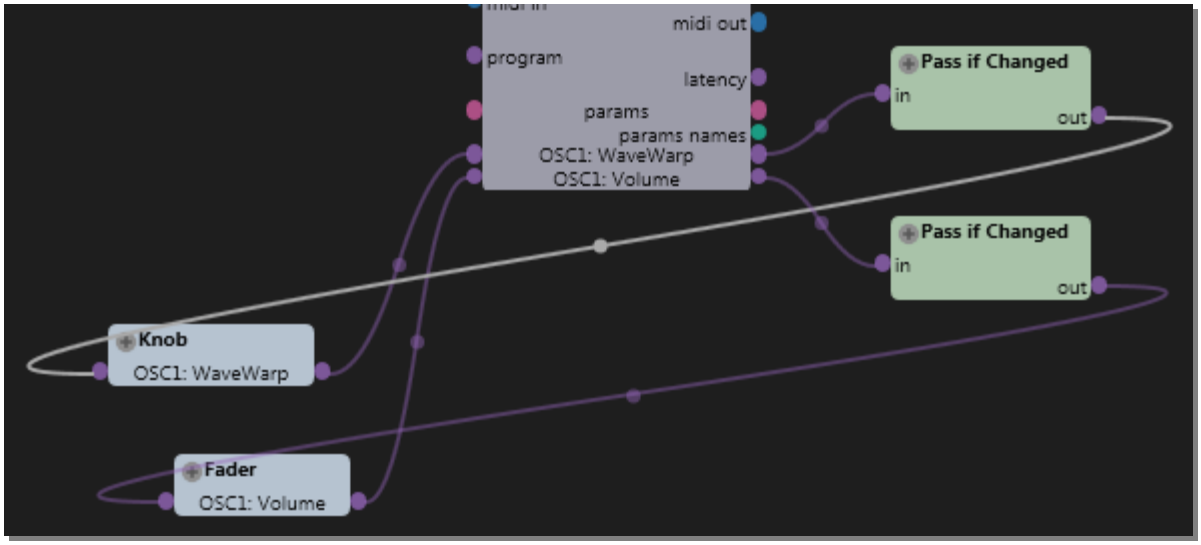
8. Plug-ins Patches

Simple interface

Once you have created some controls for a specific plug in, like here with some knobs and faders, those controls become a kind of master object for the plug in. This means that you can't edit them anymore in the plug-in GUI and moreover, if you load a preset you'll not have a correct feedback of the new values.



A good solution is to create a feedback from the parameter outlets, using a *pass only if change* module to obtain:



As reminder, *the pass if change* module passes the input value only when it has changed. Otherwise, nothing passes through the module, **as if the wire was deleted**.

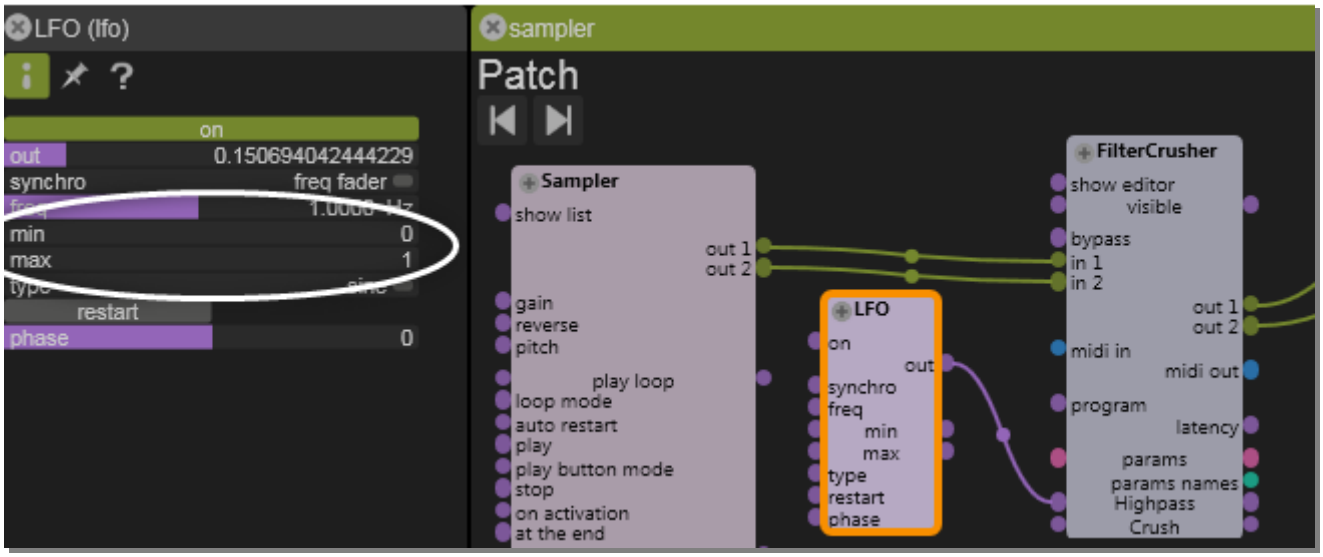
If you change the preset in the plugin GUI, Usine's knobs will be updated.

9. Transforming Data's

Change scale

Following the idea of transforming a flux to another one, here is a chapter which illustrates the scale transformation of values.

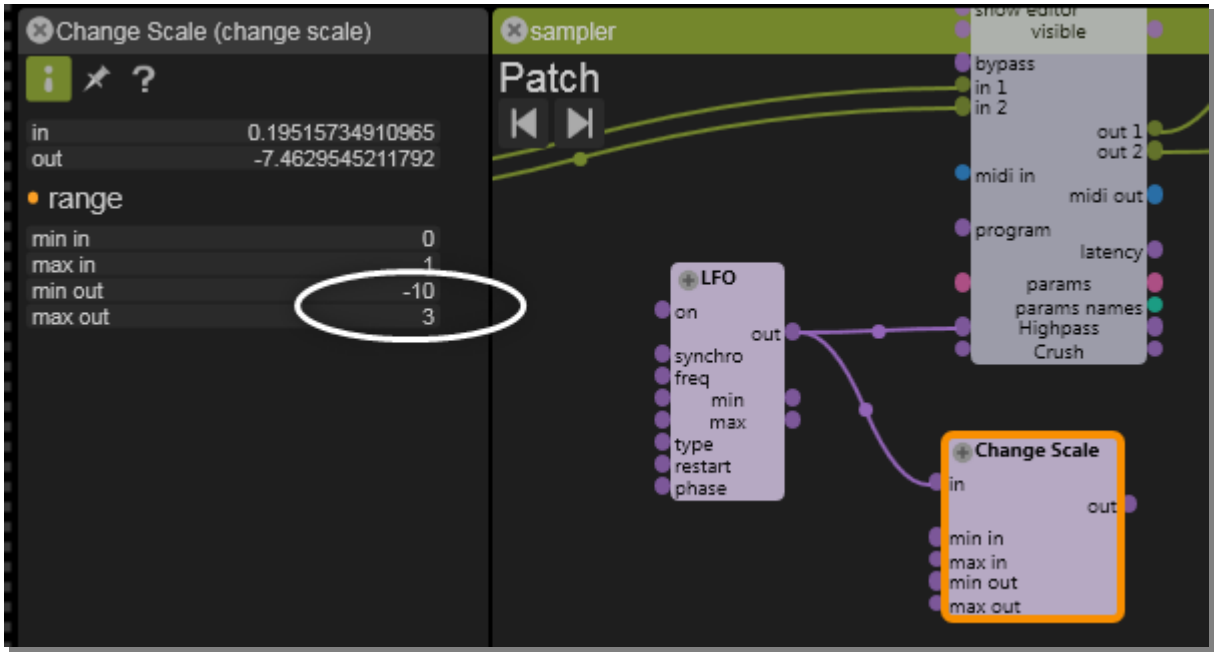
An LFO module is used to control the highpass value of the filtercrusher Plug-in with values in the range [0..1].



We want to use this *LFO* module to create a kind of tremolo effect using the *gain* inlet of the *sampler* module.

The main problem is, while the LFO module sends values into a range of 0 to 1, (linked to a VST parameter), the *gain* inlet is waiting for -∞ to +12dB values.

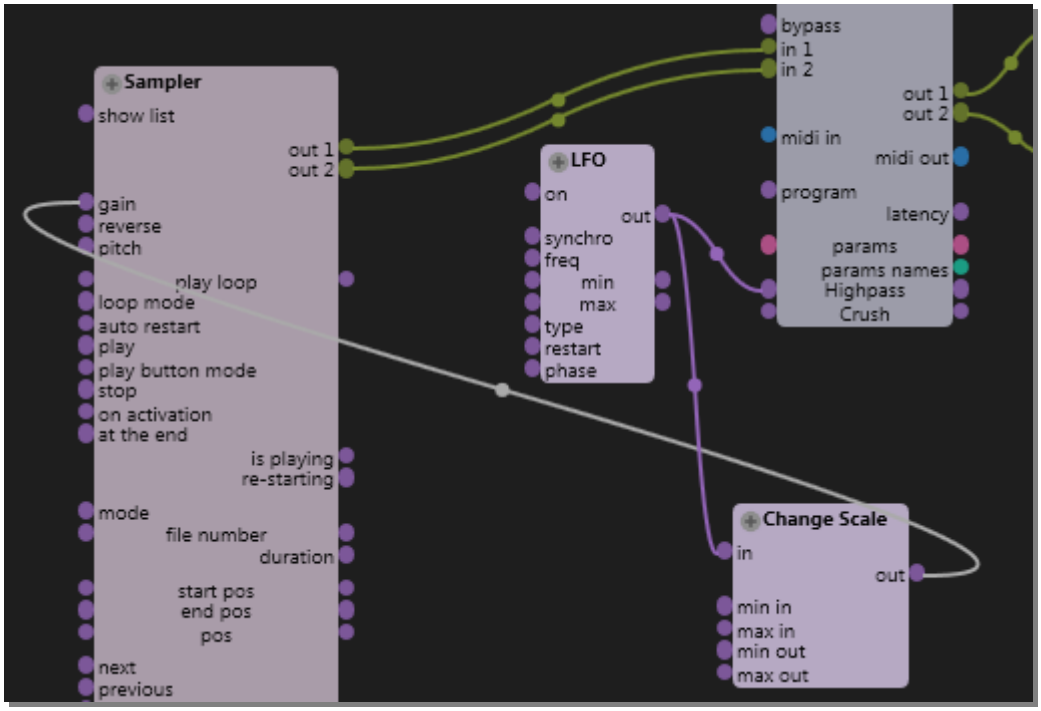
So drag & drop directly a *change scale* module in the patch:



Connect its input to the *LFO* and its output to the *gain* inlet of the sampler

This module let us define new minimum and maximum output range, respectively *min out* and *max out.*

For an interesting musical result, we can use -12 to 0 values.



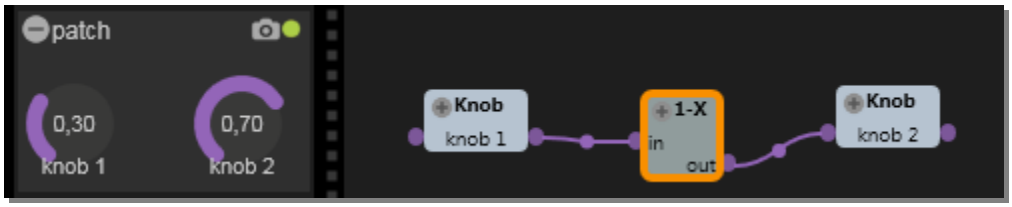
Now you can choose a good *LFO* frequency value to get the best modulation speed.

Values inversion

Often it's interesting to know how to inverse values. That's quite easy when values are in the range 0...1 value. The *1 - X* math module is made for that.

Formula: 1-(knob value)

For example, Knob is sending 0.2 so we have: 1-0.2 =0.8, then 0.3 so 1-0.3 = 0.7 and so… the range is inverted.



On any other ranges values, use the *change scale* module seen above.

10. Sub Patches

General concept

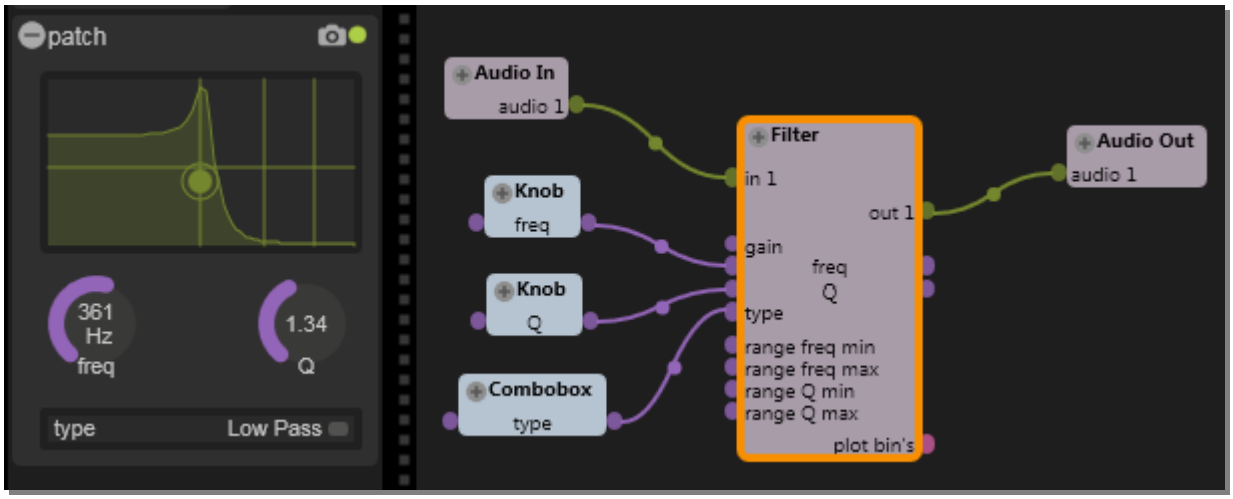
Usine has a complete hierarchic patch organization. Each patch can contain other patches called sub-patches. Sub-patches can contain themselves, other sub-sub-patches, etc.

There is no difference between patch and sub-patch except that a sub-patch may have data *input/output* modules.

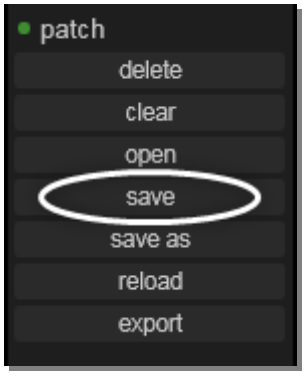
You will understand this new concept with an example: imagine that you need a mono audio filter, while Usine's library has only stereo ones.

Drag & drop the *Filter audio (LP, HP, BP, and NOTCH)* module in an empty patch and choose *mono* in the query window.

Create the *audio input* and *audio output* modules, and 2 knobs, one for frequency and another for the Q parameter:



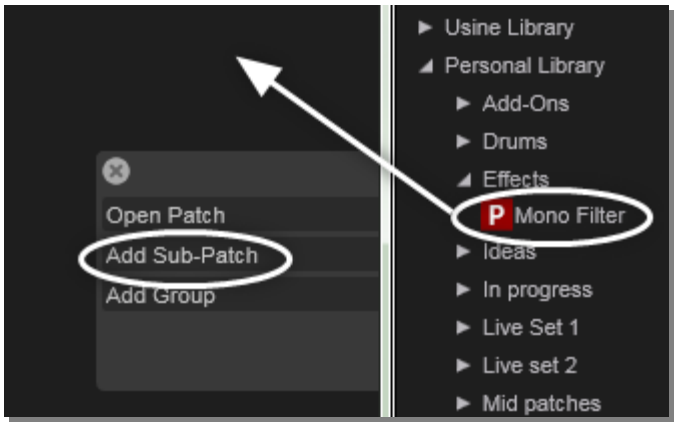
Now, save this patch; use the contextual menu/save patch (or `[CTRL/CMD ALT+S]`). Save it in your personal library folder with a name like "Mono Filter.pat".



Now you are ready to use your sub-patch.

Use the menu Patch/Clear Patch to create a new empty patch.

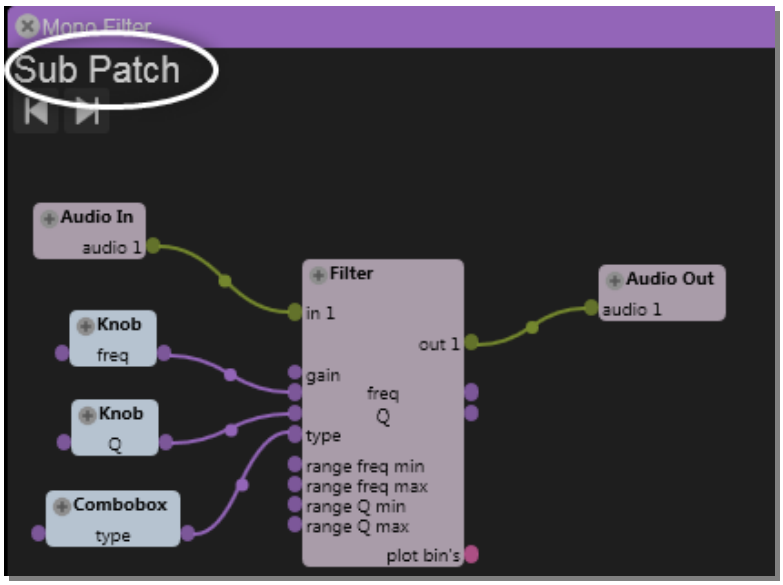
Click on *Browser/Library,* and the previous patch should appear as "Mono filter.pat" (if not, click on rescan). Select it, and drag & drop it. Choose *Add* Sub-Patch in the menu that appears.



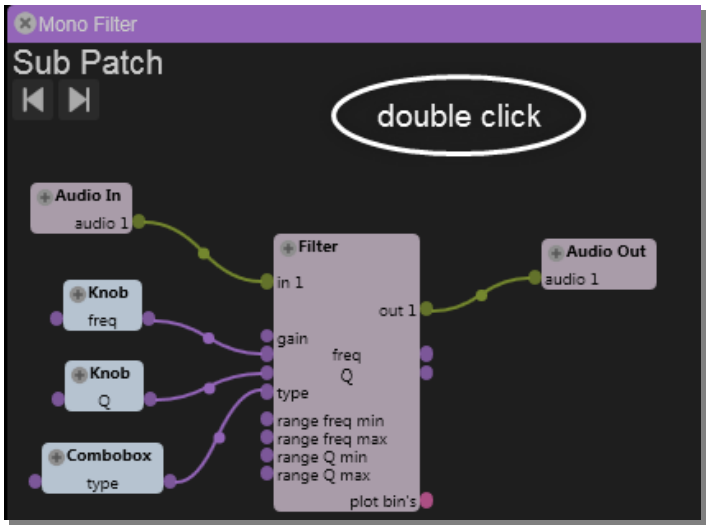
The Mono filter sub-patch has been created on the main patch.

Double click on the sub-patch to display its structure.

Here the "Sub-patch" in the top left corner as a reminder.



Double click on an empty area of the sub-patch to come back to its *parent* patch.



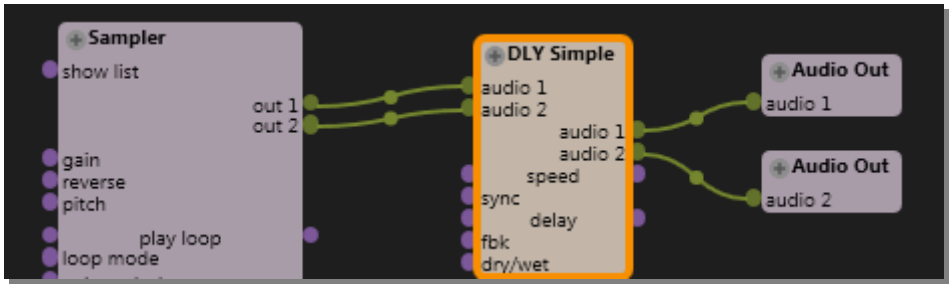
Add inlets and outlets to sub Patches

Case study

Add a knob in my main patch that communicates with a sub-patch.

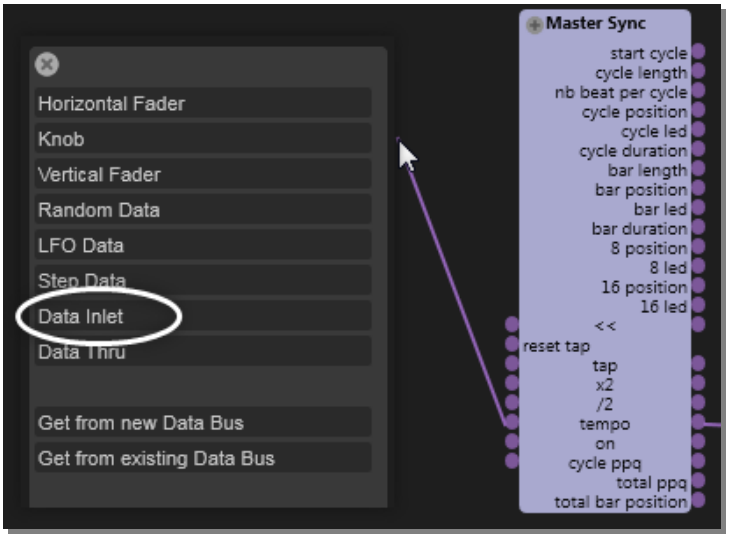
Method

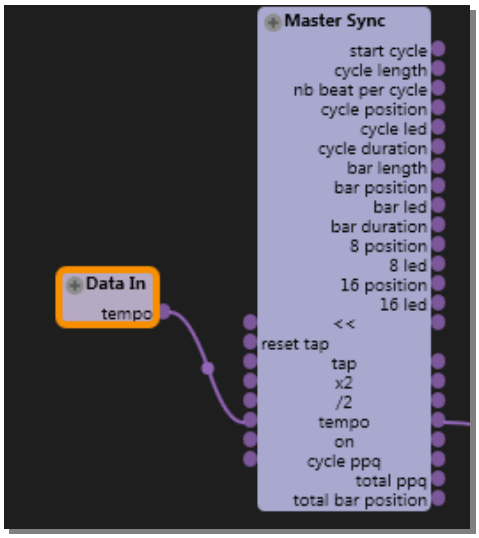
Consider a simple patch made of a sampler and a sub-patch delay (*Browser/Library/Delays*):



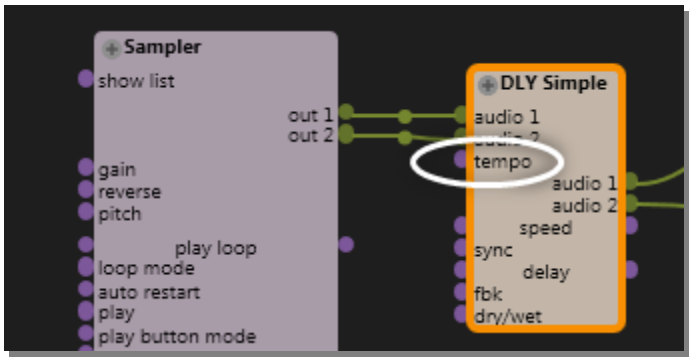
By double clicking on the sub-patch (to open it), you can see it has a module *master synchro*.

Drag a wire from the *tempo* inlet and choose the menu that appears to add a *data inlet* module:

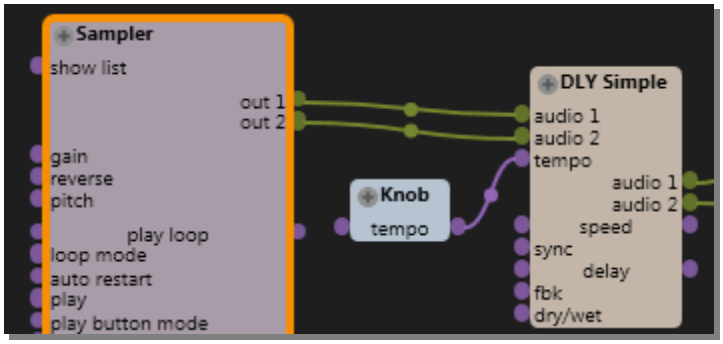




If we return to the main patch, a new inlet *tempo* has been created for the sub-patch:



Now, we only have to create the *knob* from the new Inlet:



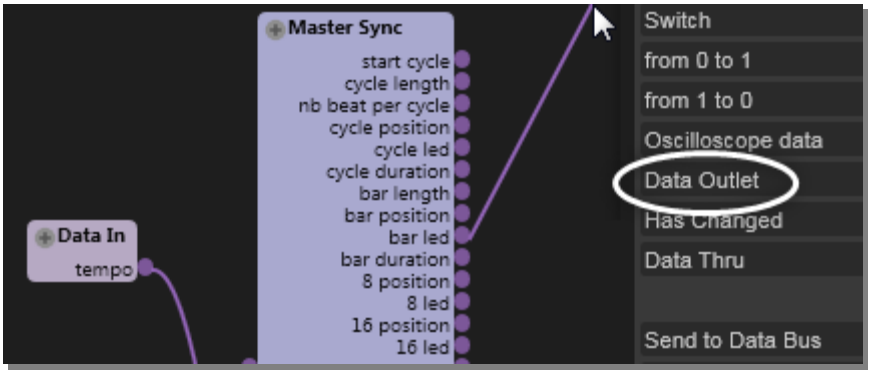
Data out module

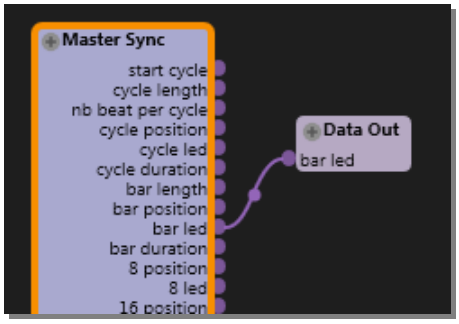
Case study

I want a led in my main patch that shows up on every beat of the Bar.

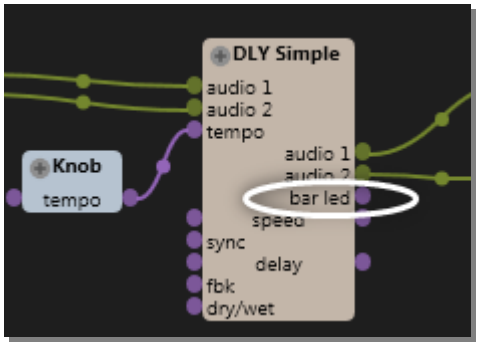
Method

Open the new sub-patch *delay* again and drag a wire from the outlet *bar led* of the *master sync* module. Then select *data outlet* in the menu that appears.



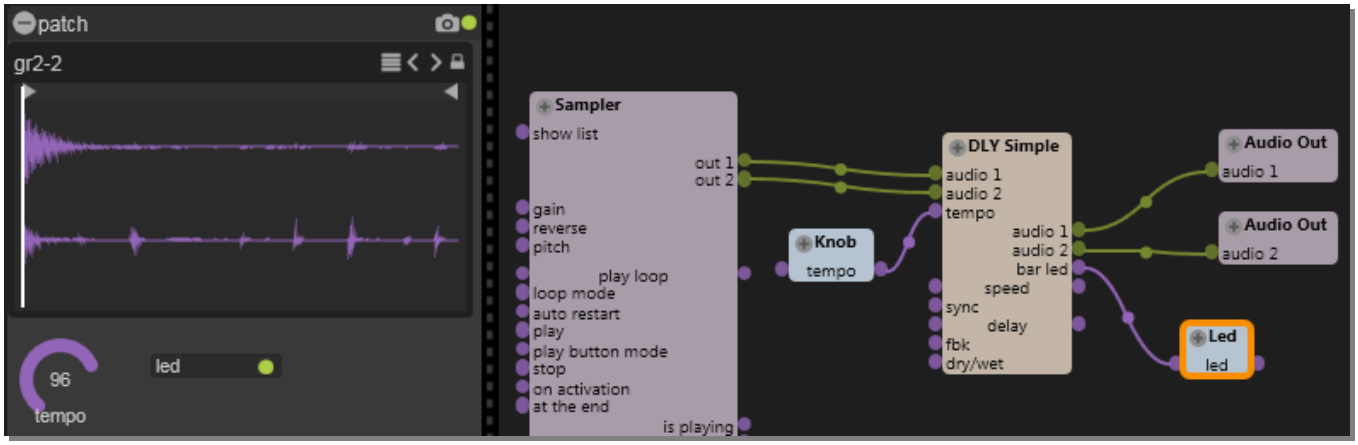


A new outlet *bar led* has been created:



Finally drag & drop a LED module and connect it to the inlet.

We now have a led that indicates each beat of the bar in the main patch:



11. Polyphony for Sub-Patches

Usine Hollyhock provides an efficient way to handle polyphony in sub-patches. The global idea is to design a single patch and ask to Usine to duplicate it into several *clones*, called *voices* that are processed in parallel.

The polyphony concept can be used in many situations:

- create polyphonic synth
- multi-channel treatment
- any data or audio manipulation, to replace repeated patches,
- etc.

Some polyphonic sub-patches examples

4in 4out Random Delay

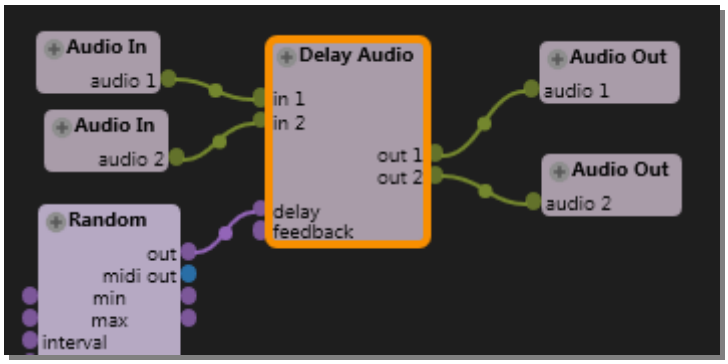
Better than hours of explanations, let's take an example: a simple 4in –> 4out random audio delay.

From the browser, drag & drop an *empty* sub-patch onto a main patch.

Go to into this sub-patch ([DBL-CLICK]), then add the following modules:

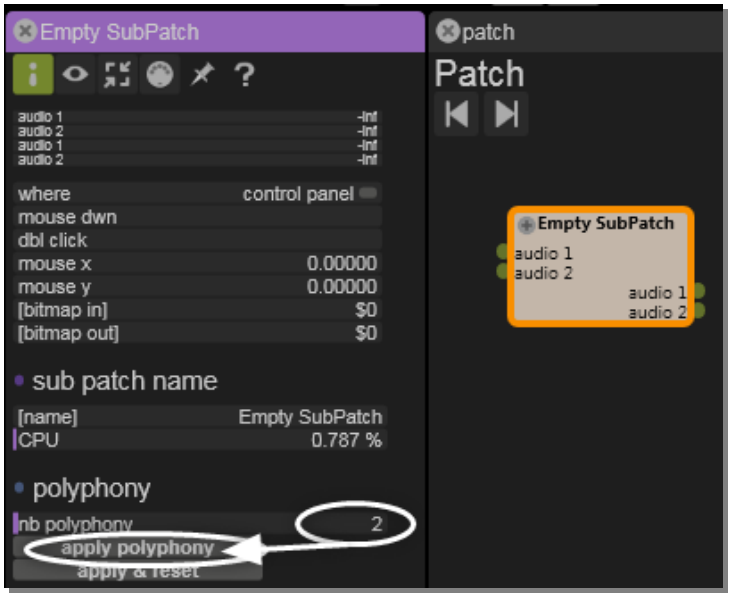
- audio inputs
- audio outputs
- audio delay (choose stereo in the query window)
- random generator

And connect them as:



This patch is stereo, and we would like to turn it into a quadraphonic one. For that, double click inside the sub-patch to go back in the parent patch and right click on the sub-patch to open its settings panel:

Change the polyphony slider (set it to 2) and click on the *apply polyphony* button.



The sub-patch has its inputs and outputs duplicated.

In other words, we have created a quadraphonic random delay by setting the polyphony of a stereo patch to 2.

In polyphonic sub-patches, the changes become active only after the *apply polyphony* button has been pressed.

4in 4out Volume

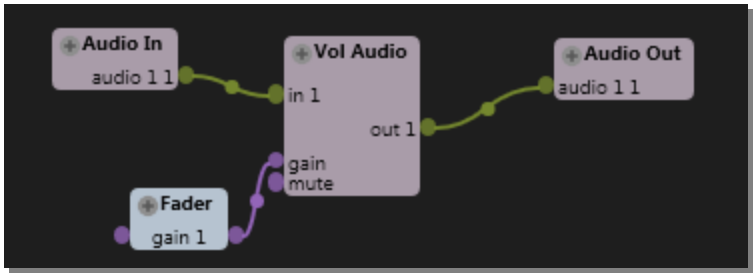
We suppose that we want to create a simple 4in 4out audio volume. This patch is actually not very interesting because this module already exists natively in Usine, but it's a good example.

1) from the browser, drag & drop an *empty* sub-patch onto a main patch.

2) go to into the sub-patch (`[DBL-CLICK]`), then add the following modules

- audio inputs
- audio outputs
- mono audio volume (choose mono in the query window)
- gain fader

And connect them as:



3) set the polyphony slider to 4,

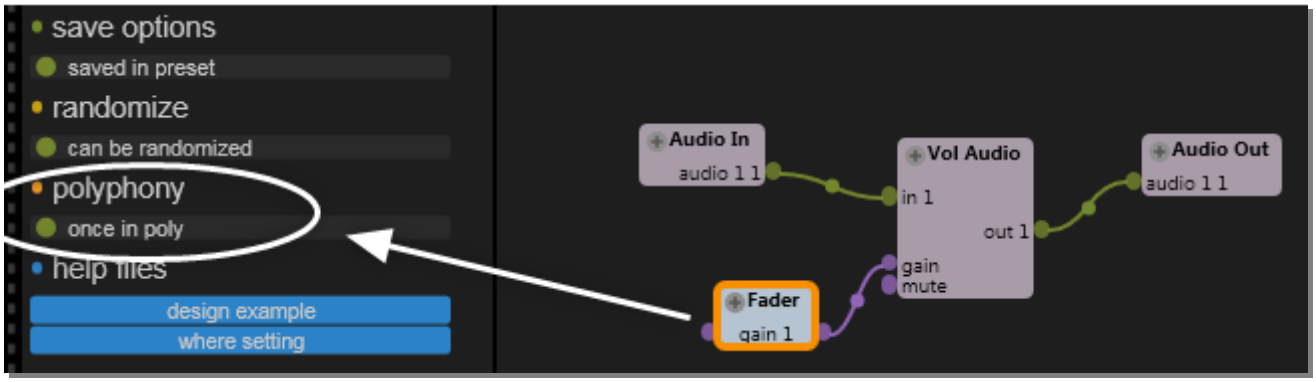
4) click on the apply polyphony button.

Now, if we comeback into the parent patch we can see something like

Input, output, and the gain fader have been duplicated 4 times.

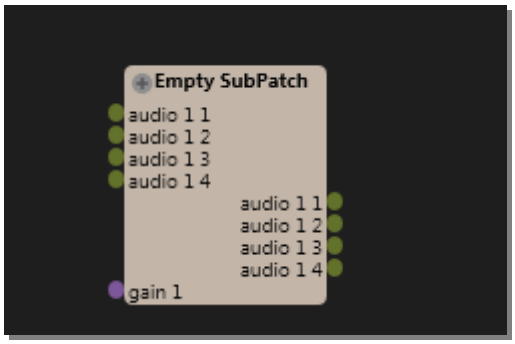
It can be interesting to have the *gain* fader duplicated 4 times, but in our case, it's better to have only one *gain* fader which controls the 4 channels at the same time?

For that, select the *gain* fader module in order to display its settings panel and turn the *once in poly* **ON**.



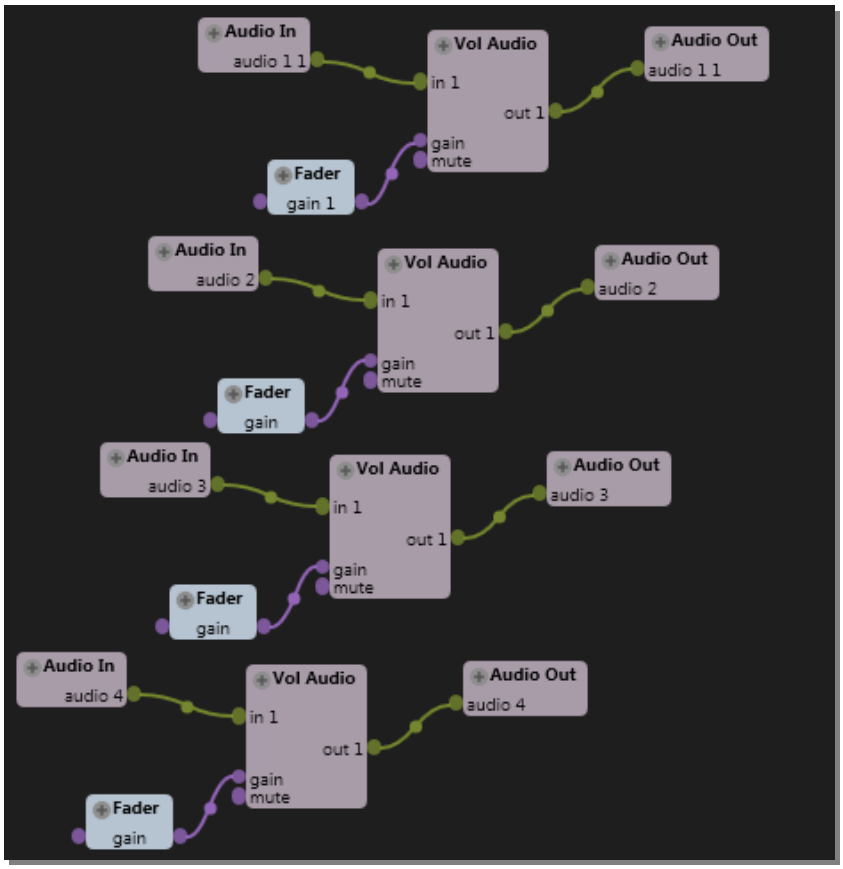
This switch asks to Usine to not duplicate the control according the polyphony. The gain fader will be created only once and will control all the *volume/gain* parameters.

After that, as usual, click on the apply polyphony button. Now in the parent patch the sub-patch looks like:

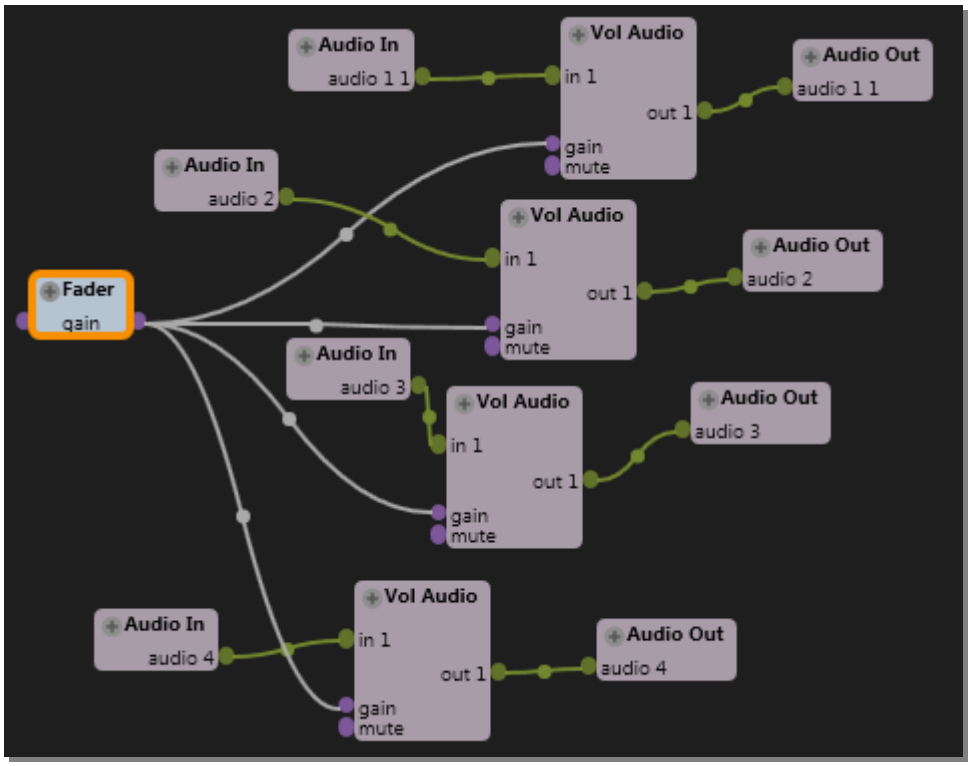


When we apply a polyphony=4, internally the patch is duplicated 4 times.

It means that the patch is equivalent to:

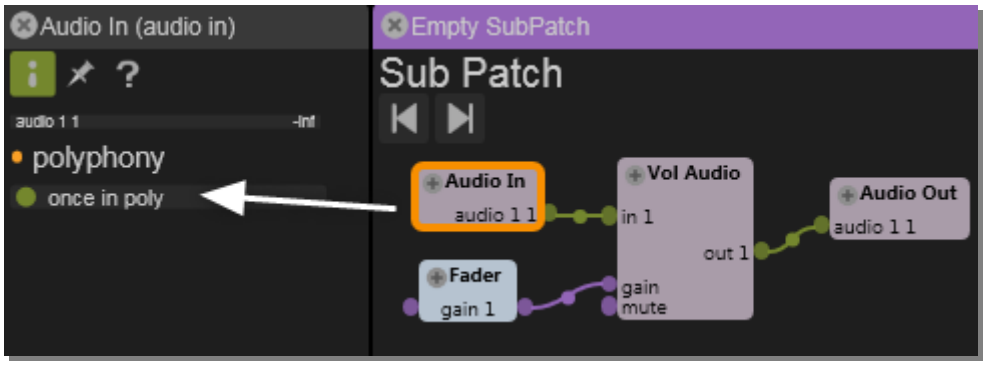


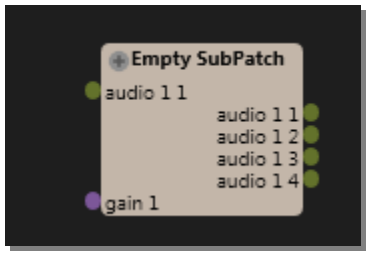
But if we turn the *one in poly* of the gain fader **ON** it becomes equivalent to:



1in/4outputs Audio

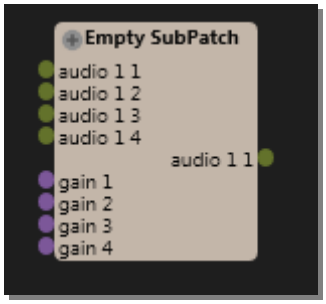
The *once in poly* option can be activated for *audio inputs* modules. For example with a mono input → 4 outputs volume.





4 in 1 out Audio mixer

The *once in poly* option can be activated for *audio outputs* modules. Here is an audio mixer where only the *audio output* module has the option *once in poly* activated:



Advanced Polyphony

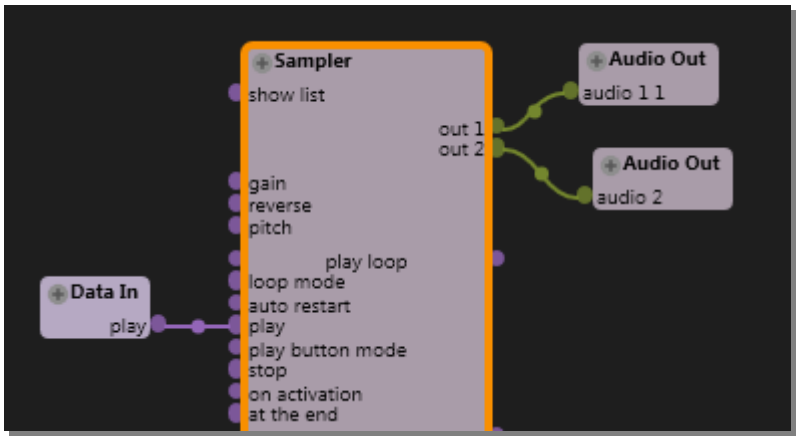
Usine can provide more powerful and complex polyphony managements with two special modules:

- *Polyphony*: gives the total polyphony of the sub patch.
- *Current voice number*: gives the current polyphony voice number.

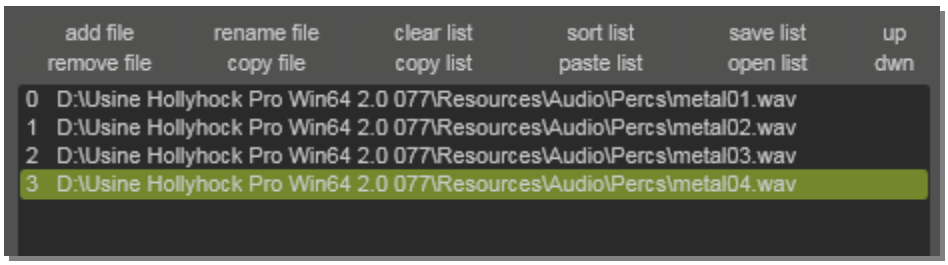
For example, we want to create a patch with 4 sample players playing 4 different samples.

- the *polyphony voice* 1 plays the sample 0,
- the *polyphony voice* 2 plays the sample 1,
- Etc...

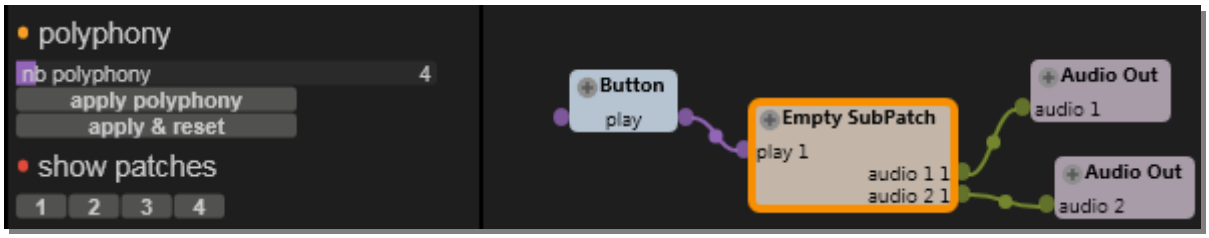
Let's create the following sub-patch. The *data in* and *audio outputs* modules are set to *once in poly.*



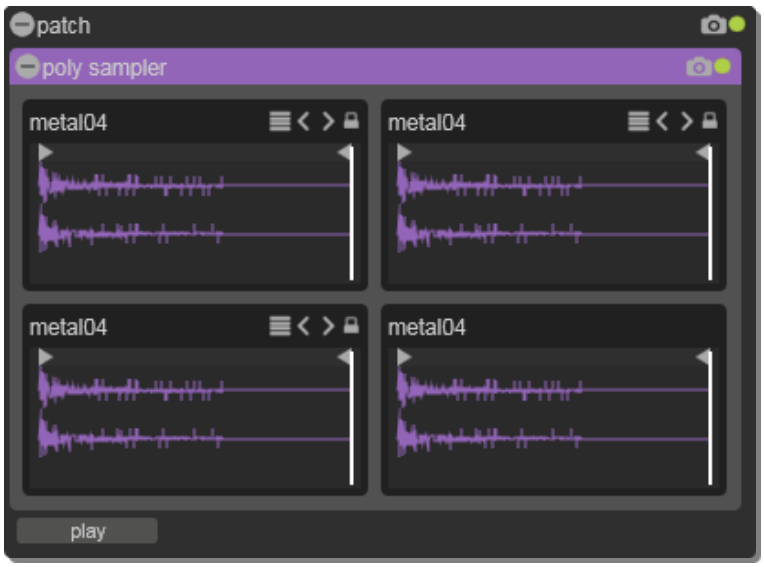
Double click on the sampler module to add 4 wave files (here *metal* 1 to *metal 4*):



Add a play button, two audio outs, then select the sub-patch and set the polyphony=4. Finally, click on the *apply and reset* button.

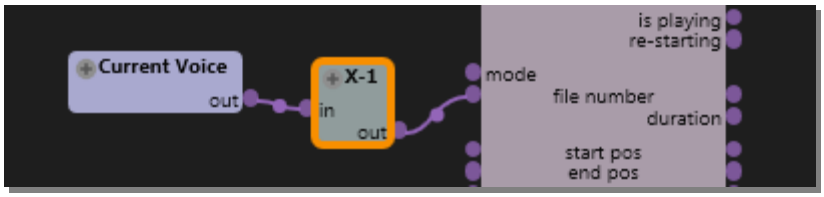


If you rearrange the *wave display* positions on the control panel, it can look like:



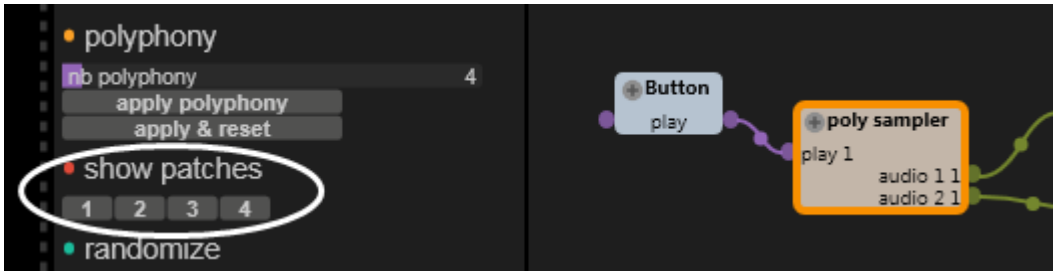
Now, the only problem is that all samplers are playing the same file…

In order to tell that the sampler-voice1 has to play the sample 0, the sampler-voice2 has to play the sample 1, etc. We can use the browser/interface control/*Current voice number* module patched like:



Don't forget to click the apply polyphony again.

You can verify yourself, looking into each *patch voice* separately, by clicking on the voices *buttons*. Then you can see that the voice 1 is playing the sample 0 and son on:



Apply polyphony and apply and reset

When you make some changes in the sub patches, you'll need to apply the polyphony again. There's a difference in the way Usine recreates the sub-patch polyphony when using this two buttons

Apply the polyphony

Applies the polyphony and try to keep the most of the previous settings (layouts, samples and so...)

Apply and reset

Resets all settings and recreates the polyphonic sub-patch from scratch.

It's more or less the same action than when you create the polyphony the first time: samplers have all the same sample, layout are all in the same place etc.

This button will also rebuild all the objects ID's.

Remember that each patch shared on the local network use its own ID (identification number). Useful especially If you experience of patches that don't send information to the good target,

So to resume, **apply polyphony** when you made minor changes **apply and reset** when you want rebuilt all from scratch

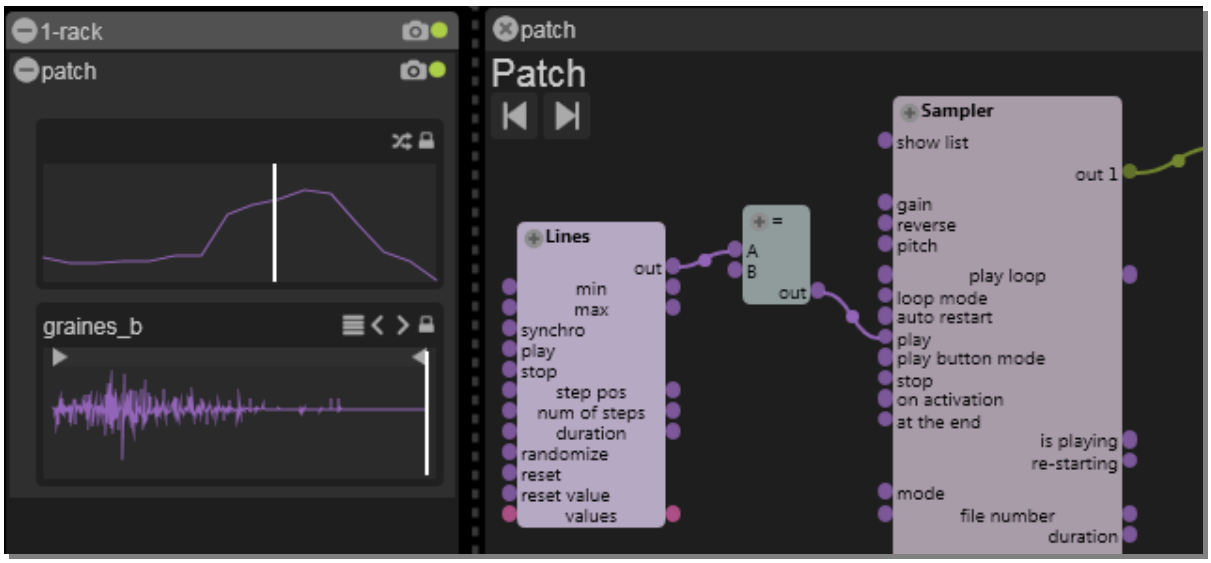
12. Math Modules

Maths modules, even the simplest ones can often be useful in patch processing.

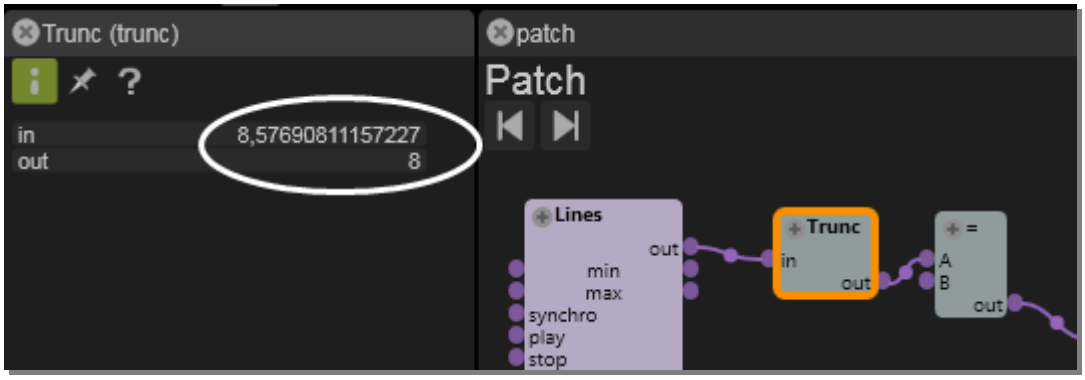
Arithmetic modules

Trunc / Round

We have a patch where the module *sequenced lines* send values between 0 and 16, and we want that when the value 8 is reached (with the equal module) the sampler play. The problem is that the Lines module sends float values, not integers. So it doesn't work.



Adding a *trunc* module will transform theses float values into integers and the patch is working now.

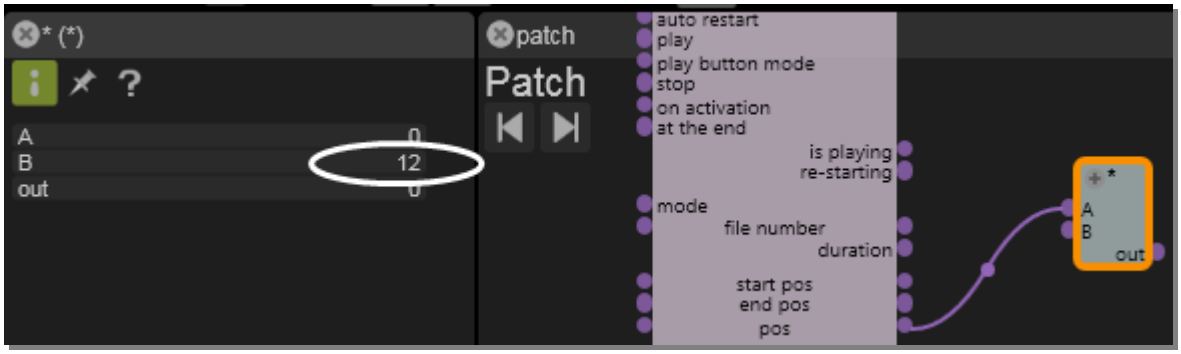


The same behaviour can be obtained with the *round* module

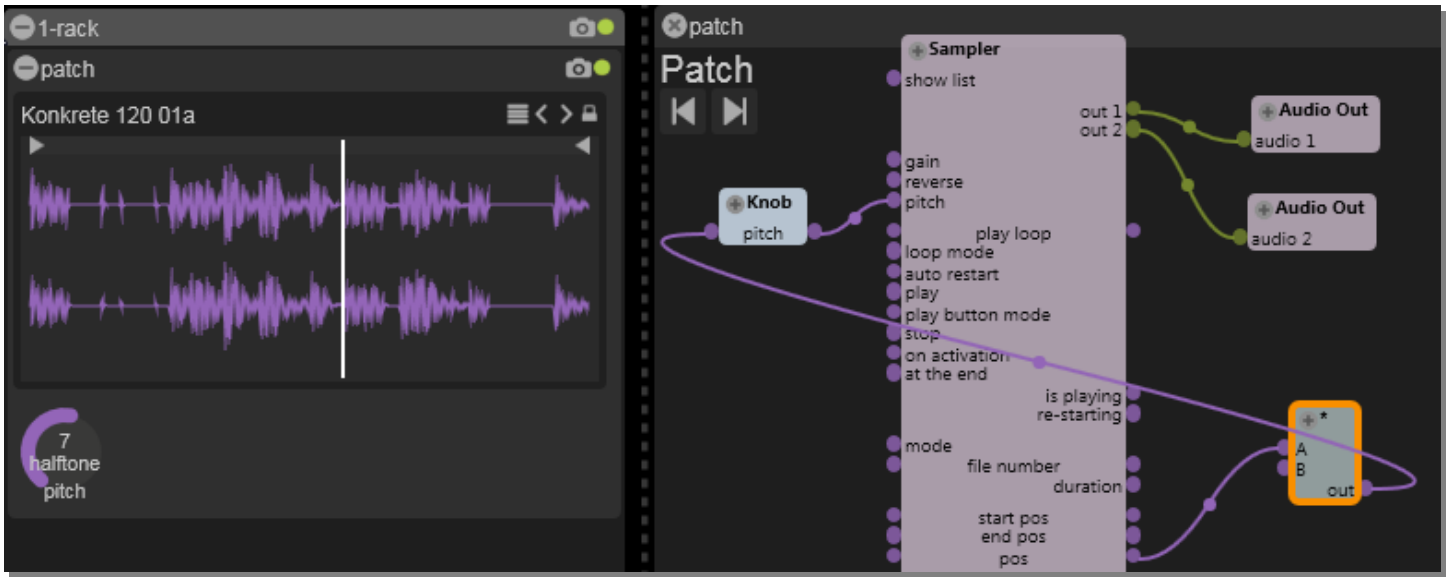
Multiplication

Now we're going to transform the value of a sampler *pos* outlet.

Just drag & drop a *A * B* module from the browser/modules/maths tab and sets its B value to 12:



So, we can connect the *A * B* module to the *pos* outlet (which changes from 0 to 1) and then to the *pitch* inlet of the sampler.



Now we have a progressive pitch effect, which will increase from 0 to 12 half tones following the playing position of the *sampler*.

Logic modules

Usine provides also logic modules.

In patches, we often face a problem like: "if **X** does this, then I would like **Y** do that". Logic modules are made for that.

Logic *module* principle states: if the condition is achieved the module sends a 1 value 0 otherwise.

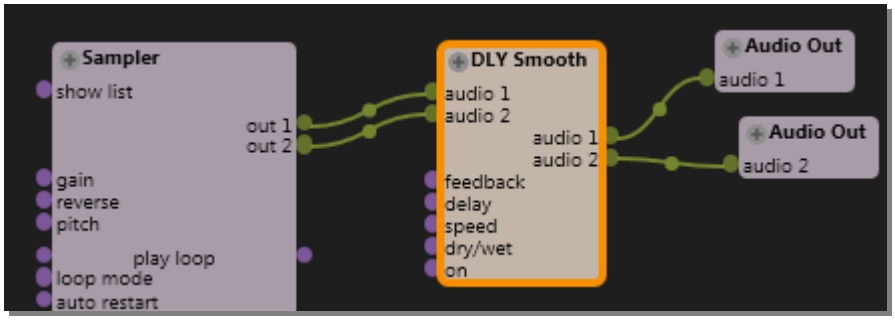
Greater or Equal (>=)

Case study

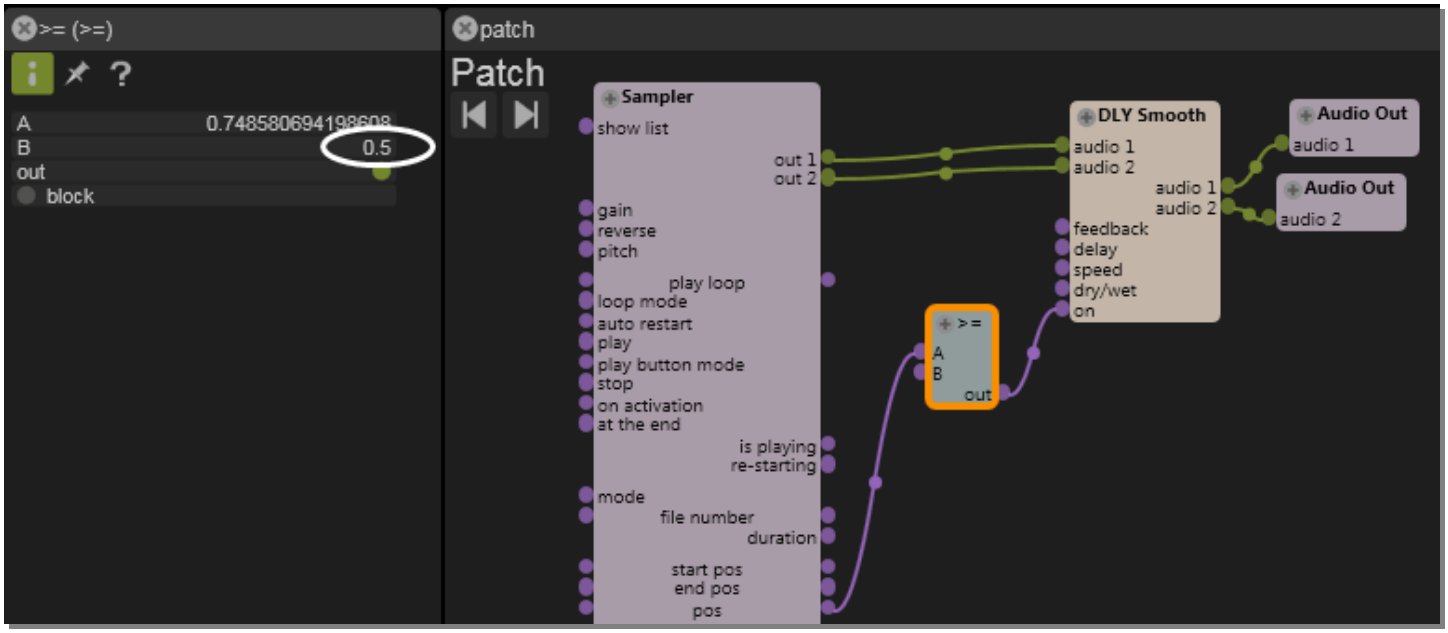
If my sampler position is greater or is equal to X, then a delay effect is turned ON.

Method

At first, add a *delay* sub-patch to the audio outlets of the sampler module. Choose the DLY *smooth* sub-patch.



Now, add a *greater or equal* module *(A>=B)* that you will find in the browser/modules/maths/logic and connect it as follow:



We can now set the B value to 0,5 for example. Now when position will reach half of the sample the *delay* sub-patch will be turned ON.

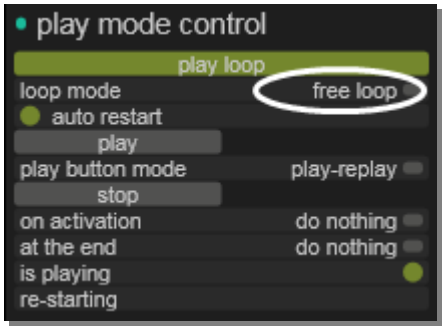
NOT

Case study

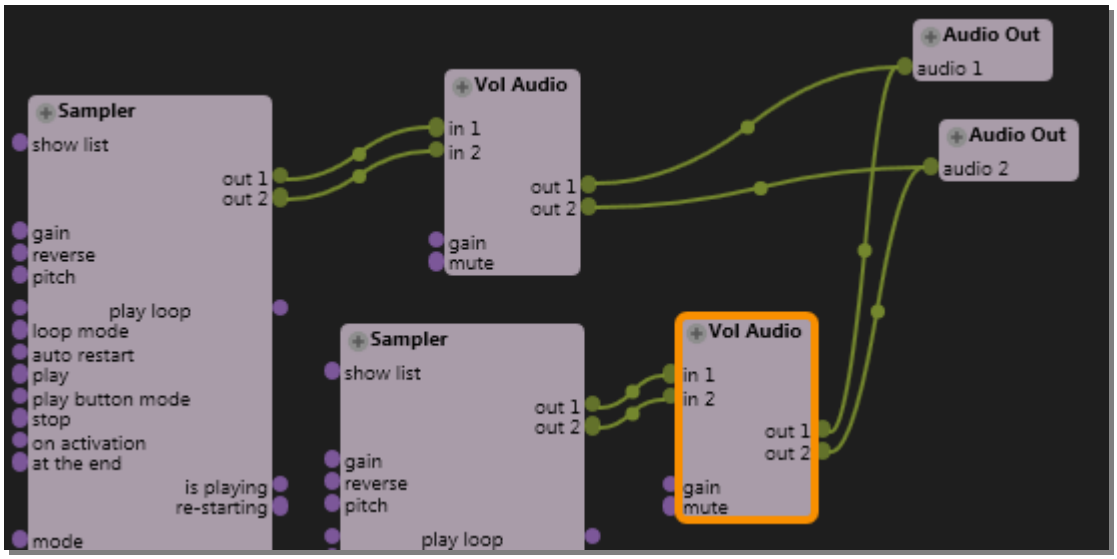
I would like to use only one switch to alternate between 2 samplers.

Method

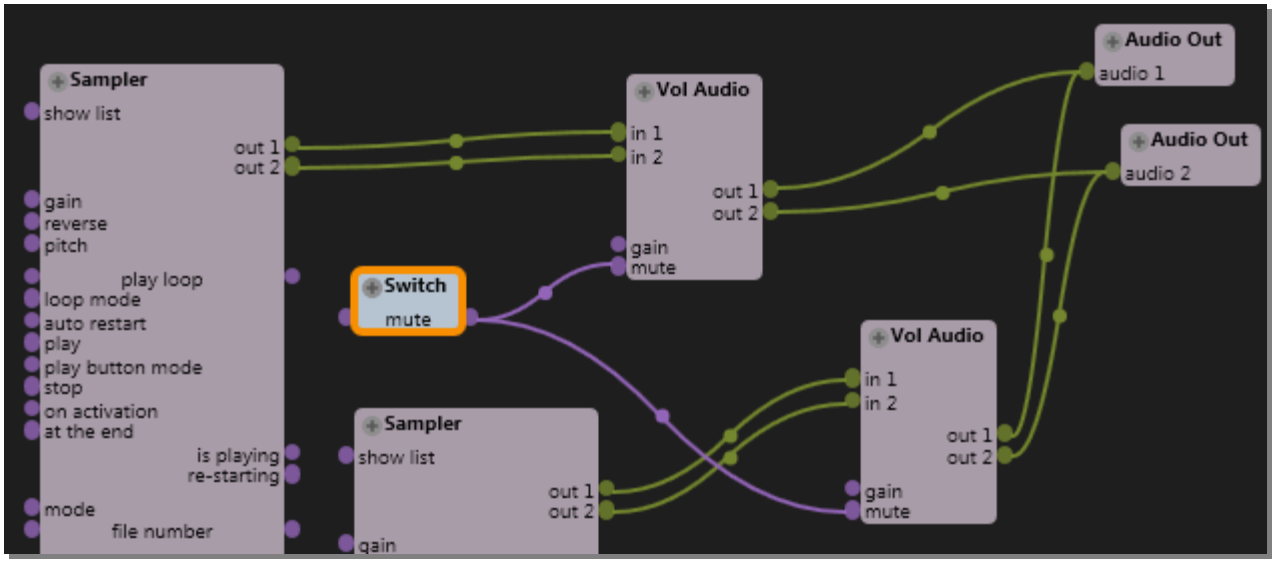
We use two sampler modules. Please check that the synchronisation mode is set to *free loop*:



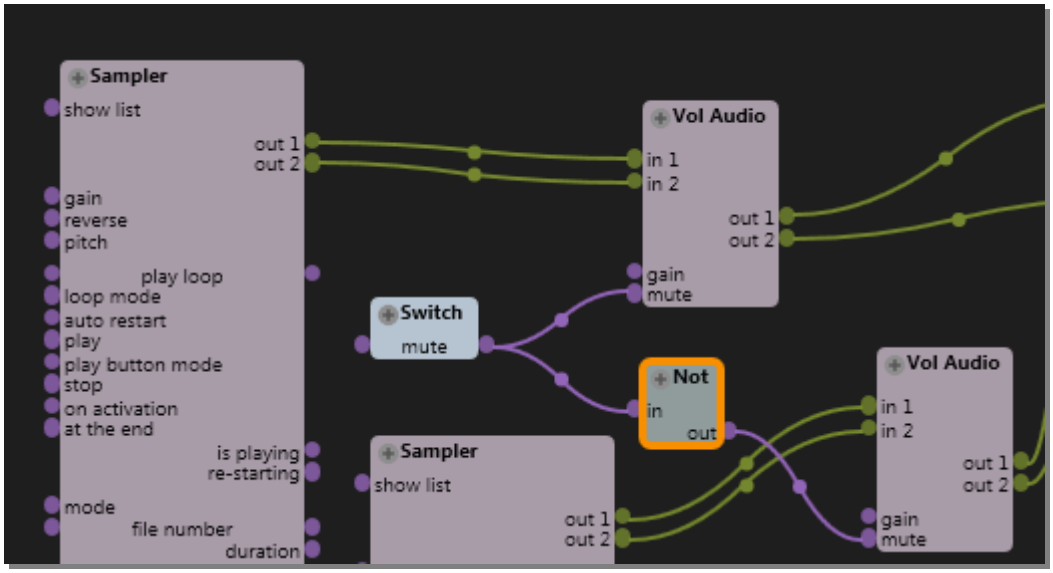
Now drag & drop a *volume/mute* module over each wire connected to the outs of the two *samplers* to obtain something like:



Add now a switch module on the *mute* inlets *volume/mute* modules to obtain:



Finally we have to inverse one of the flux by adding a *not A* module.



The *not A* module is useful for inverting 0-1 values:

- if in=0 then out=1
- if in=1 then out=0

Note: You can set different names to the *on* and *off* states of the switch.

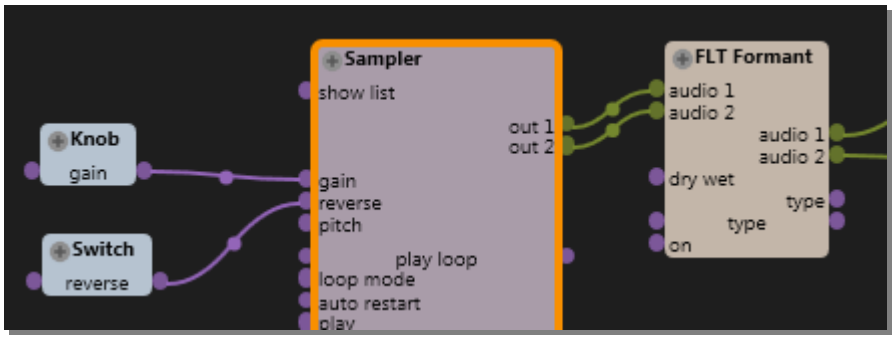
AND

Case study

I've got a filter sub-patch connected to a sampler. I would like to activate it only when the volume gets over -10 dB and while the reverse switch is ON

Method

We connect a Library/filter/ FLT Formant sub-patch to the sampler:

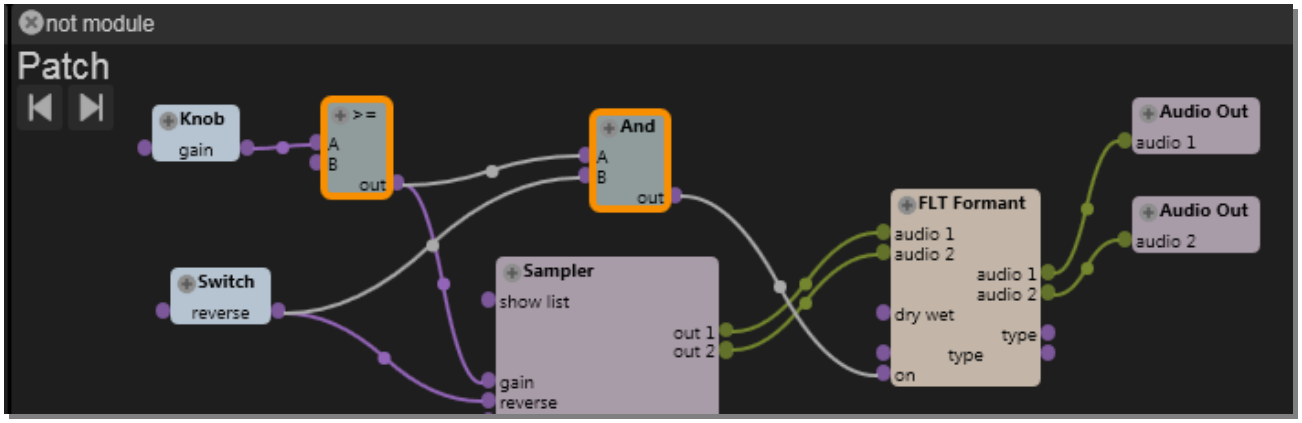


Now, add two logic modules, a *greater or equal (>=)* then a *and*.

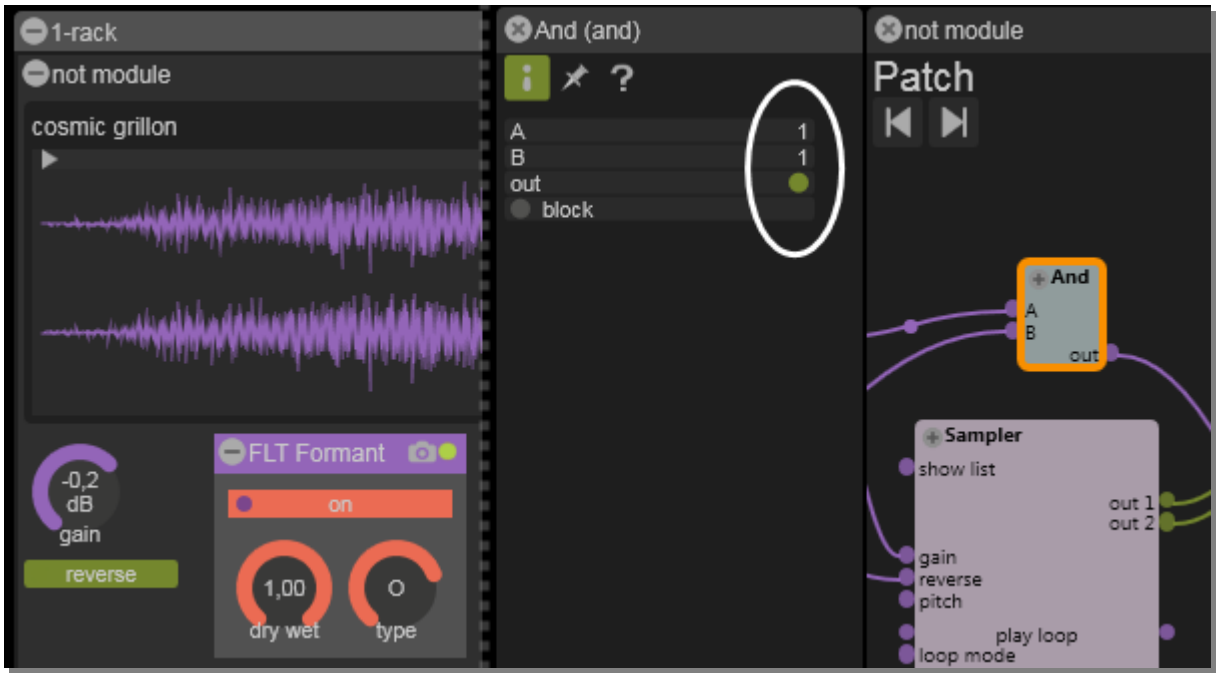
Take the *gain* outlet of the *knob* and link it to the A inlet of the *A>=B* module.

The B value is set to -10.

The *A>=B* module is linked to the A inlet of the *A and B* module and the switch *reverse* outlet is linked to the B inlet to obtain:



If the gain is over -10 and the *reverse* switch is *on*, the two conditions match, so the *A and B* module sends the 1 value to the *on* switch of the *filter* sub-patch.



13. Switches Groups

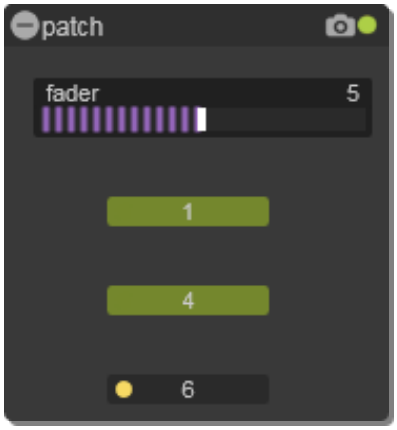
Case study

I want 3 switches with different values to control fader, but only one can be activated at the same time.

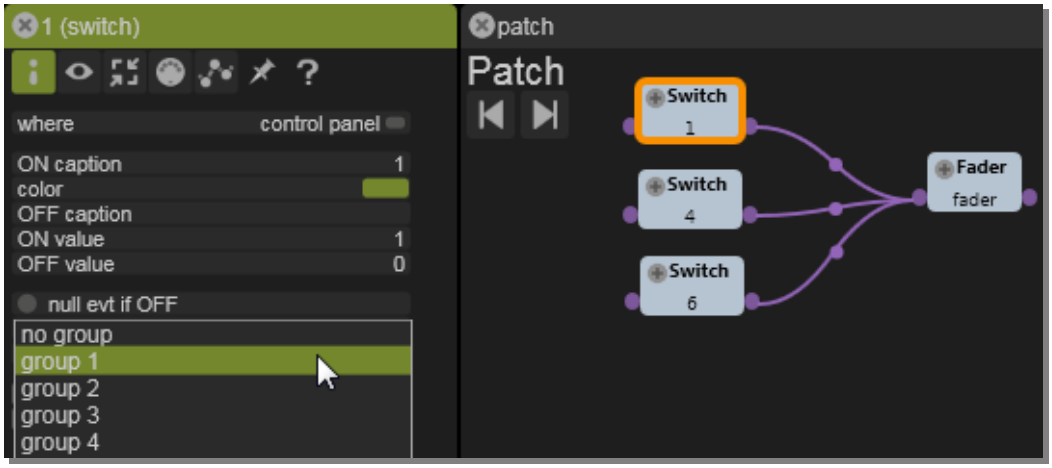
Method

Create a simple *patch*: 3 switches named respectively 1, 4, 6. Assign them different ON values, respectively 1, 4, 6.

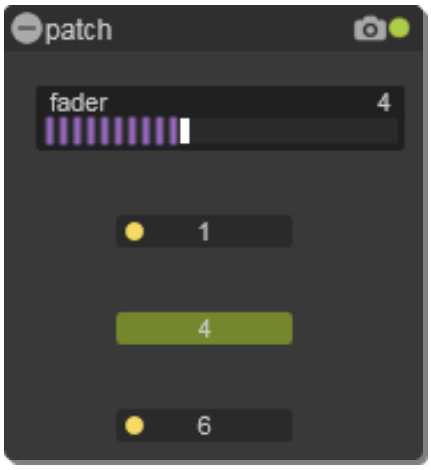
The first problem is that you could unintentionally activate 2 or even the 3 switches at the same time. In this case the values will be added… It's not really what we want.



Assign to all switches a group, here we choose the *group 1*:

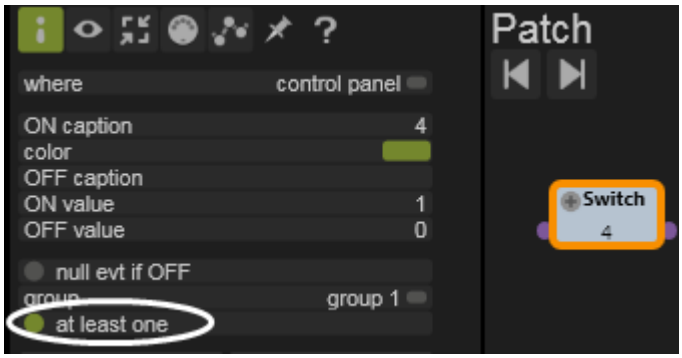


Now when you set a switch ON all the others are set to OFF.



But if you set it off you'll have the 0 value.

If you don't need this option, activate the at *least one* option activated on all switches.



14. Control's Location

Usine lets you display modules at different places.

Container

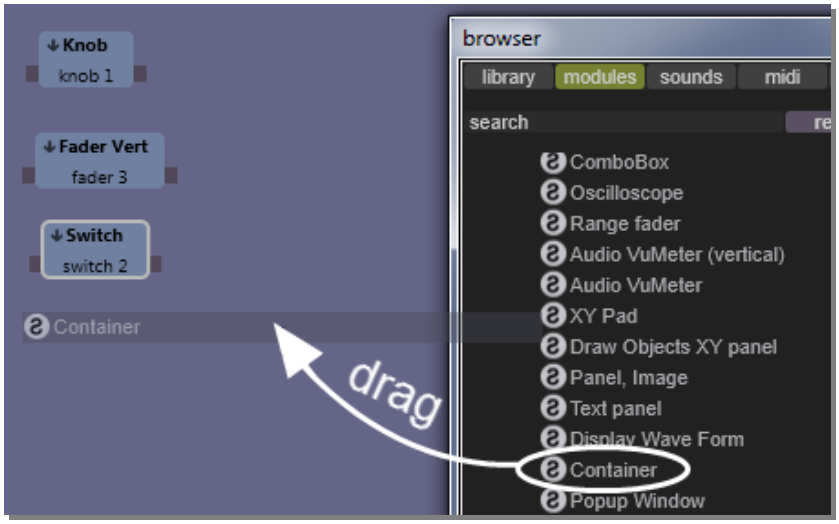
Duplicate sets of controls

Case study

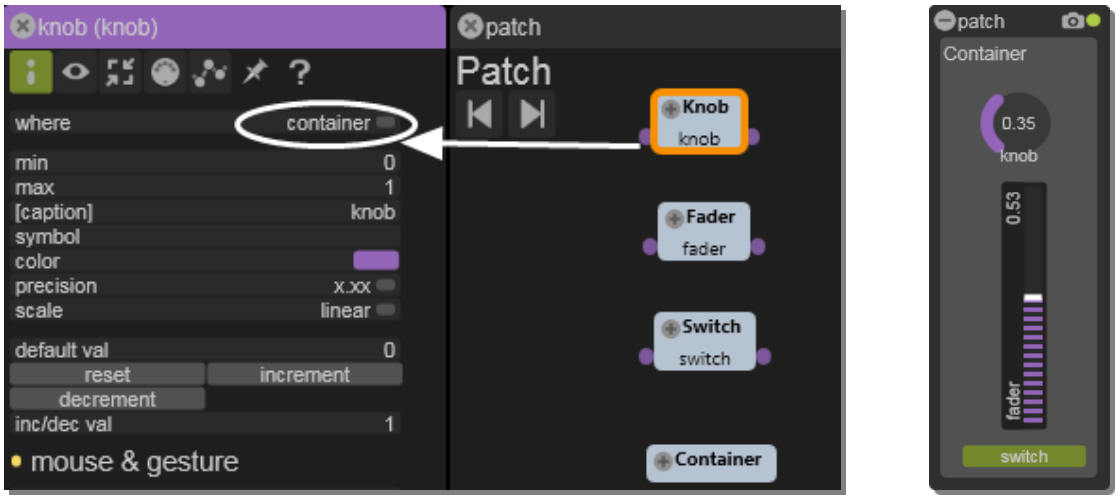
We need a switch, a fader and a knob and all of these duplicated.

Method

It is easy to copy and paste these items, but it will sometimes be tedious to align and move them without breaking the organization. The *container* will simplify layout and design. It creates a group of objects that can be easily copied and moved. Just drag & drop a *container* module into the patch:



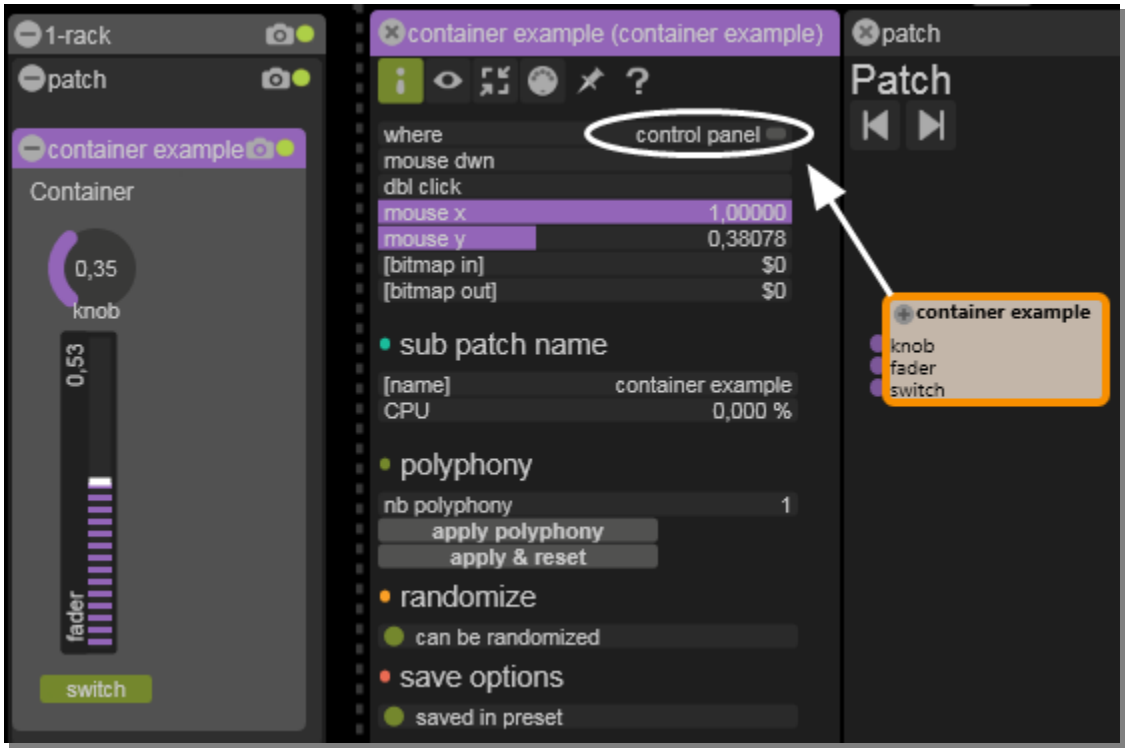
Once the *container* is in place, we can change the visibility option (where) of each element so they appear in the *container*:



Now save this new patch in your personal library.

You can now add this new patch, named for example *container example* from the library. Be careful to drag it as a sub-patch:

Now click on the sub-patch and change its *where* option to *control panel*. In this way it will be visible in the control panel.



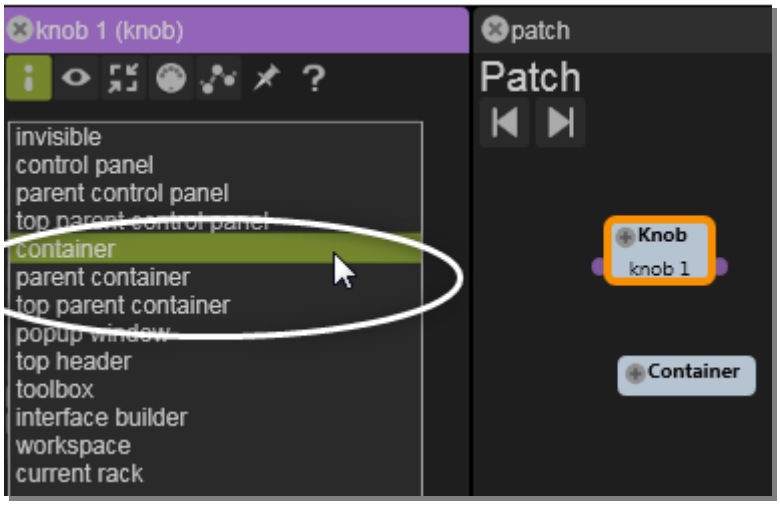
It will be easy to duplicate the sub-patch and to align those copies (Lock/Unlock menu or [CTRL+L]). But be sure to click only on the background of the container when moving it, otherwise, you take the risk to move some other controls.

Hierarchy of container

You can put one container per patch or sub-patch or sub-sub-patch, etc.

A control placed, for example, in a sub-sub-patch can be visible in:

1. the container of the sub-sub-patch itself (container) ,
2. the container present in the sub-patch parent (parent container),
3. the container of the main patch (top parent container).

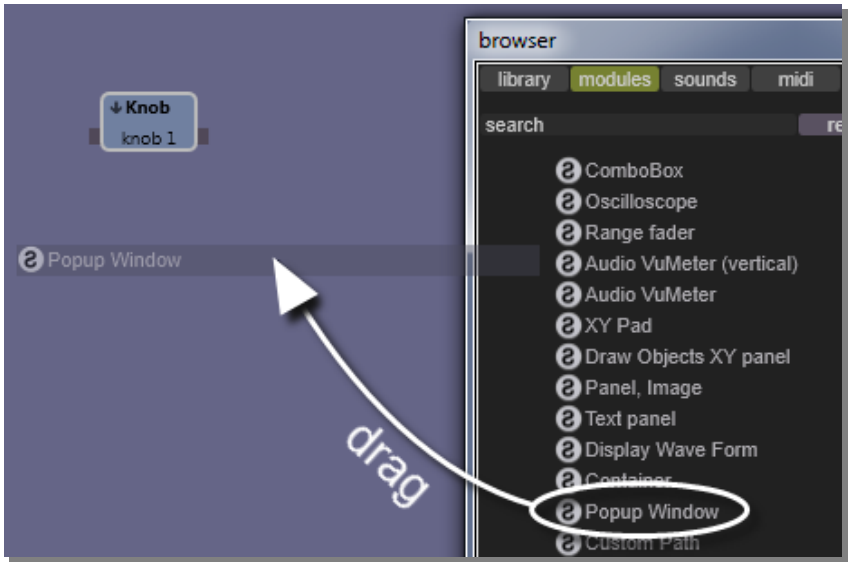


If no container is present in a sub-patch controls will be located in the first available container in the parent patches according the hierarchy.

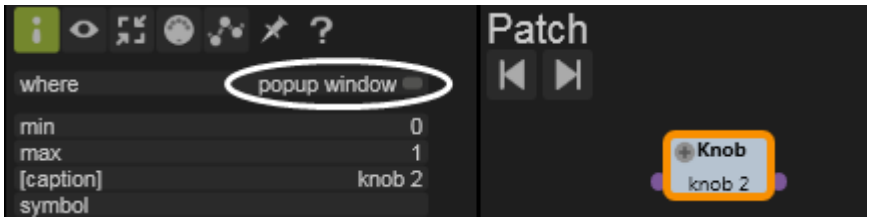
Pop-up Window

As its name suggests, this module will bring up a popup window to display some elements.

Before selecting this location, you'll need to drag & drop a *popup window* module from the */modules/interface design/* tab of the browser:



Select *pop-up window* in the *where* option for the objects you need in this window



By default the popup module can be opened by clicking on the toothed wheel icon of the tool bar.

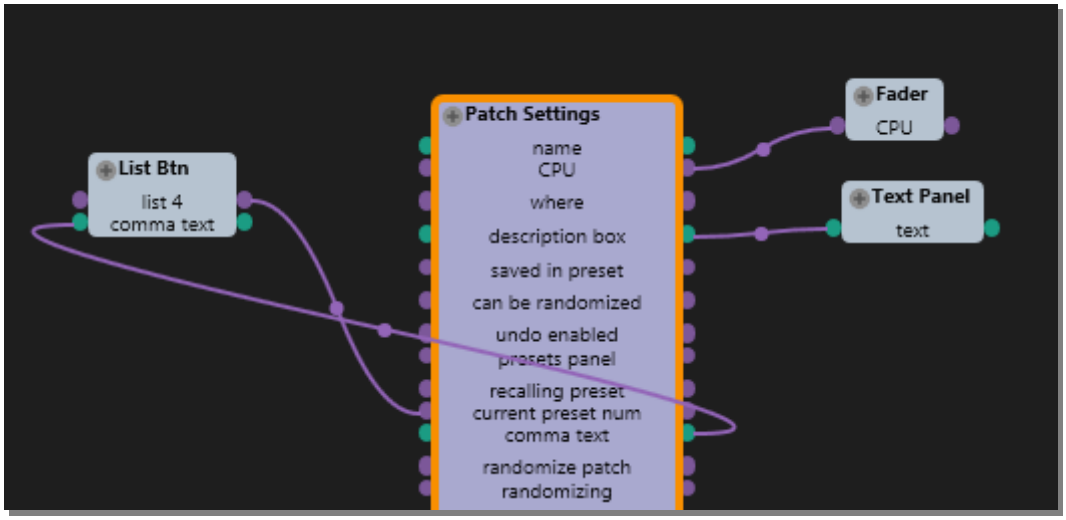
15. Usine's Panels

The Usine panel modules will let you interact directly with parameters and Usine functions.

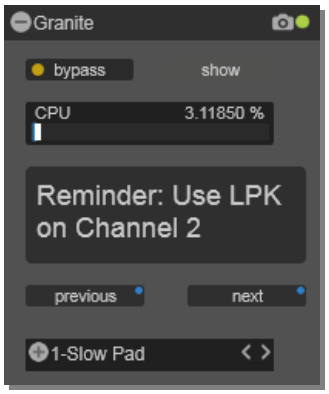
Usine becomes very powerful when you have access to any parameter, command, and settings within each patch.

For example, the *patch settings* module can give you access to the integrated preset manager.

- A List box to display presets,
- A fader to display the CPU load of the patch,
- A text panel module to display the description box.



Moreover the *listbox* can be set to invisible and replaced by two *buttons* to navigate in the presets, eventually driven by MIDI learn. Finally the control panel looks like:

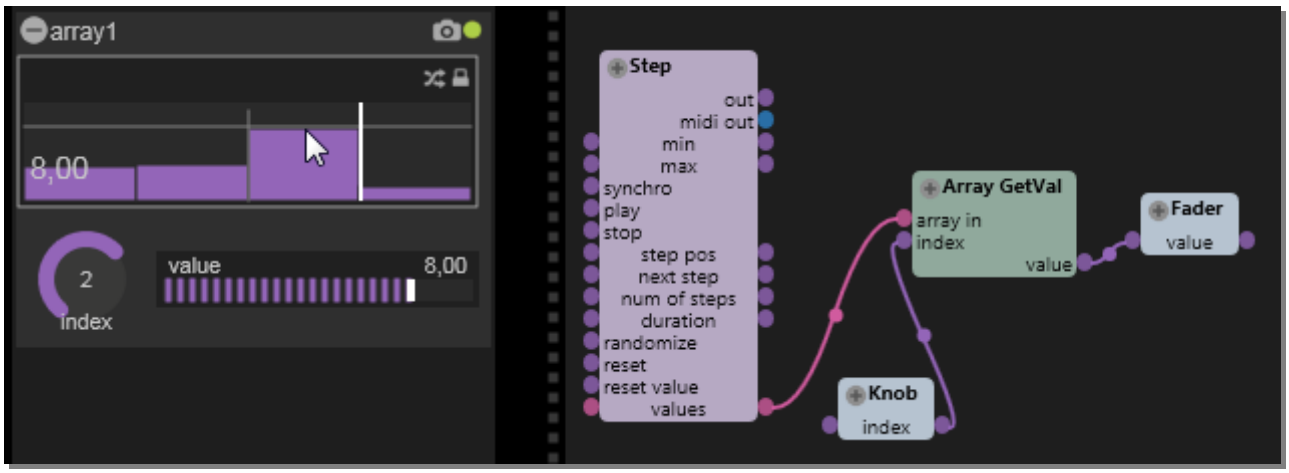


16. Array's

To understand *array* concept, you can imagine a series of values like in a step sequencer, each value has a specific index: an index always start with 0, so first value has the index 0, second value the index 1

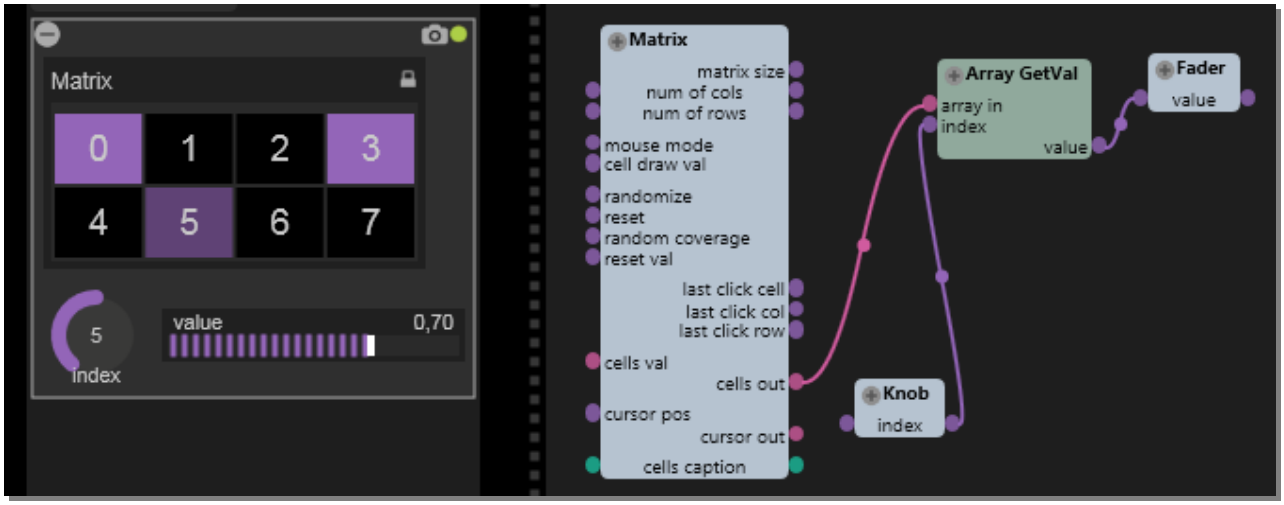
Now you can, for example, extract a value at a precise index by using the *get array element value* module.

In the following example, we retrieve the array value at the index 2:



The same operation can be achieved with a *matrix*, even if data's are organized on several rows.

In the following example, we retrieve the value of the cell number 5.



17. Improving the CPU Load

Sometimes, patches and workspace can be improved, on CPU side, with small decisions. Let see some tips to reduce the CPU consumption if needed.

Simplification

Simplification is the key. Try to always have this question in mind when patching: *Do I really need all of these features? Do I need all of these effects in the same patch?*

Polyphony

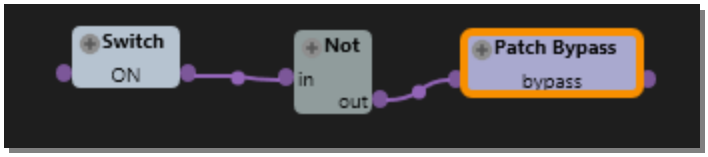
Polyphonic sub-patches provide improved CPU load over multiple iterations of objects. If you have repetition in your patch design, think about using sub-patches to simplify and improve the CPU load.

Patch ON/OFF

The *patch on off* module turns the current patch ON or OFF. In polyphonic sub-patches, this is the best way to disable voices if they are not played.

Patch bypass

The *bypass* module can also be useful to switch OFF a patch when it is not needed. The *bypass* module works in the same way that the *Patch ON OFF* module does. The difference however, is that when the patch is bypassed the audio and MIDI flows are still passed through.



18. Conclusion

Starting a project

* Gather the first modules or sub-patches you need and place them in an empty patch.

* Notice the available inlets/outlets of modules.

* Look into the library and add-ons for a patch approximating the idea of what you want to achieve.

* Observe, isolate some parts to understand how they work.

* The old method: create a diagram on paper may also help clarify an idea?

Troubleshooting Problems

Simplify the patch or divide it into smaller pieces to isolate the problem, even copy and paste parts into empty patches to analyse them more freely.

Consider using the *oscilloscope*, the *led* and the *trace* module to analyse/verify the exact inflow.

Good patching!

And remember that the forum is made to share ideas and can help when needed. There will always be someone willing to help you.